
common-ovf-tool Documentation

Release 1.9.1

the COT project developers

February 21, 2017

1	Introduction	1
1.1	Examples	1
2	Installing COT	3
2.1	System requirements	3
2.2	Installing COT using <code>pip</code>	3
2.3	Installing COT from source	4
2.4	Installing helper programs	4
3	Using COT	7
3.1	Getting CLI help	7
3.2	Verifying and installing helper programs with <code>cot install-helpers</code>	8
3.3	Inspecting OVF contents with <code>cot info</code>	10
3.4	Updating product information with <code>cot edit-product</code>	10
3.5	Adding disks to an OVF with <code>cot add-disk</code>	11
3.6	Packaging additional files into an OVF with <code>cot add-file</code>	12
3.7	Removing files from an OVF with <code>cot remove-file</code>	12
3.8	Customizing hardware profiles with <code>cot edit-hardware</code>	13
3.9	Customizing OVF environment settings with <code>cot edit-properties</code>	15
3.10	Embedding bootstrap configuration with <code>cot inject-config</code>	16
3.11	Deploying an OVF to create a VM with <code>cot deploy</code>	17
3.12	Creating a VM on VMware vCenter/vSphere with <code>cot deploy esxi</code>	18
4	Glossary	21
5	Change Log	23
5.1	1.9.1 - 2017-02-21	23
5.2	1.9.0 - 2017-02-13	23
5.3	1.8.2 - 2017-01-18	23
5.4	1.8.1 - 2016-11-12	24
5.5	1.8.0 - 2016-11-08	24
5.6	1.7.4 - 2016-09-21	24
5.7	1.7.3 - 2016-09-06	25
5.8	1.7.2 - 2016-08-17	25
5.9	1.7.1 - 2016-08-12	25
5.10	1.7.0 - 2016-08-05	25
5.11	1.6.1 - 2016-07-07	26
5.12	1.6.0 - 2016-06-30	26

5.13	1.5.2 - 2016-06-17	26
5.14	1.5.1 - 2016-06-07	26
5.15	1.5.0 - 2016-06-06	27
5.16	1.4.2 - 2016-05-11	27
5.17	1.4.1 - 2015-09-02	28
5.18	1.4.0 - 2015-09-01	28
5.19	1.3.3 - 2015-07-02	28
5.20	1.3.2 - 2015-04-09	29
5.21	1.3.1 - 2015-04-09	29
5.22	1.3.0 - 2015-03-27	29
5.23	1.2.4 - 2015-03-06	29
5.24	1.2.3 - 2015-02-19	30
5.25	1.2.2 - 2015-02-19	30
5.26	1.2.1 - 2015-02-03	30
5.27	1.2.0 - 2015-01-16	30
5.28	1.1.6 - 2015-01-05	31
5.29	1.1.5 - 2014-11-25	31
5.30	1.1.4 - 2014-11-12	31
5.31	1.1.3 - 2014-10-01	31
5.32	1.1.2 - 2014-09-24	32
5.33	1.1.1 - 2014-08-19	32
5.34	1.1.0 - 2014-07-29	32
5.35	1.0.0 - 2014-06-27	32
6	Contributing to COT	33
6.1	Follow coding guidelines	33
6.2	Add automated unit tests	34
6.3	Update documentation	35
6.4	Add yourself as a contributor	35
6.5	Open a pull request	35
7	Credits	37
8	COT package reference	39
8.1	Virtual machine definition modules	39
8.2	Command modules	51
8.3	Helper library modules	71
8.4	User interface modules	91
8.5	Sub-packages	97
9	COT.disks package reference	129
9.1	API	129
9.2	Disk modules	129
10	Indices and tables	135
	Python Module Index	137

Introduction

COT (the Common OVF Tool) is a tool for editing [Open Virtualization Format](#) (.ovf, .ova) virtual appliances, with a focus on virtualized network appliances such as the [Cisco CSR 1000V](#) and [Cisco IOS XRv](#) platforms.

COT's capabilities include:

- Add a disk or other file to an OVF/OVA
- Edit OVF hardware information (CPUs, RAM, NICs, configuration profiles, etc.)
- Edit product description information in an OVF/OVA
- Edit OVF environment properties
- Display a descriptive summary of the contents of an OVA or OVF package
- Embed a bootstrap configuration text file into an OVF/OVA.
- Remove files and disks from an OVF or OVA package
- Deploy an OVF/OVA to an ESXi (VMware vSphere or vCenter) server to provision a new virtual machine (VM).

Examples

Displaying a summary of OVA contents:

```
> cot info --brief csr1000v-universalk9.03.17.01.S.156-1.S1-std.ova
-----
csr1000v-universalk9.03.17.01.S.156-1.S1-std.ova
COT detected platform type: Cisco CSR1000V
-----
Product:  Cisco CSR 1000V Cloud Services Router
Vendor:    Cisco Systems, Inc.
Version:   03.17.01.S.156-1.S1-std

Files and Disks:
File Size  Capacity Device
-----
  csr1000v_hddisk.vmdk      71.5 KiB      8 GiB hddisk @ SCSI 0:0
  bdeo.sh                   52.42 KiB
  README-OVF.txt            8.534 KiB
  README-BDEO.txt           6.748 KiB
  cot.tgz                   116.8 KiB
  csr1000v-universalk9.03.17.... 425 MiB      cdrom @ IDE 1:0

Hardware Variants:
```

```

System types:          vmx-08 vmx-09 vmx-10 vmx-11
                      Cisco:Internal:VMcloud-01
SCSI device types:    virtio lsilogic
Ethernet device types: VMXNET3 virtio

Configuration Profiles: CPUs      Memory NICs Serials Disks/Capacity
-----
1CPU-4GB (default)    1        4 GiB   3        2 1 /      8 GiB
2CPU-4GB              2        4 GiB   3        2 1 /      8 GiB
4CPU-4GB              4        4 GiB   3        2 1 /      8 GiB
4CPU-8GB              4        8 GiB   3        2 1 /      8 GiB

Networks:
GigabitEthernet1     "Data network 1"
GigabitEthernet2     "Data network 2"
GigabitEthernet3     "Data network 3"

Environment:
Transport types: iso

Properties:
<config-version>          "1.0"
Router Name               ""
Login Username            ""
Login Password            ""
Management Interface      "GigabitEthernet1"
Management VLAN           ""
Management Interface IPv4 Address/Mask ""
Management IPv4 Gateway   ""
Management IPv4 Network   ""
PNSC IPv4 Address         ""
PNSC Agent Local Port     ""
PNSC Shared Secret Key    ""
Remote Management IPv4 Address (optional, deprecated) ""
Enable SCP Server         "false"
Enable SSH Login and Disable Telnet Login "false"
Enable Password           ""
Domain Name               ""
License boot level        "ax"
Console                   ""
Resource template         "default"
Intercloud Mode           ""
Intercloud Mode Management Key ""
Intercloud Control Port   ""
Intercloud Tunnel Port    ""
Intercloud Tunnel Header Size "148"
Intercloud Tunnel Interface IPv4 Address ""
Intercloud Tunnel Interface Gateway IPv4 Address ""

```

Adding a custom hardware configuration profile to an OVA:

```
> cot edit-hardware csr1000v.ova --output csr1000v_custom.ova \
    --profile 1CPU-4GB --cpus 1 --memory 4GB
```

Customizing OVF environment properties:

```
> cot edit-properties csr1000v.ova --output csr1000v_custom.ova \
    --properties mgmt-ipv4-addr=10.1.1.100/24 \
    mgmt-ipv4-gateway=10.1.1.1
```

Installing COT

- *System requirements*
- *Installing COT using `pip`*
 - *Installing optional features*
- *Installing COT from source*
 - *Downloading COT*
 - *Install the COT libraries and script*
- *Installing helper programs*

System requirements

- COT requires either Python 2.7 or Python 3.3 or later.
- COT is tested to work under Mac OS X and Ubuntu Linux and similar distros.
- COT now has limited support for CentOS and Python 2.6 as well.

Installing COT using `pip`

Since COT is written in Python, it can be installed like any other Python package using the `pip` tool. For most users this is the recommended installation method.

```
sudo pip install cot
```

If you have already installed COT and wish to update to the latest available version:

```
sudo pip install --upgrade cot
```

Installing optional features

COT has a number of optional Python package dependencies that enable optional features. If you want to use these features, you can instruct `pip` to install them as part of the COT installation process, or you can install them separately after the fact.

- Tab-completion of COT CLI parameters in `bash`, enabled with the `argcomplete` package.

```
sudo pip install cot[tab-completion]
```

or

```
sudo pip install argcomplete
```

Note: After installing `argcomplete` by either method, you must configure your `bash` environment to enable it. Refer to the `argcomplete` documentation for the required steps.

Installing COT from source

If you wish to install bleeding-edge unreleased code or make code contributions of your own, you can install COT from source as described below.

Downloading COT

You can download COT via Git or using HTTP.

```
git clone git://github.com/glennmatthews/cot
cd cot
```

or

```
wget -O cot.tgz https://github.com/glennmatthews/cot/archive/master.tar.gz
tar xzf cot.tgz
cd cot-master
```

or

```
curl -o cot.tgz https://github.com/glennmatthews/cot/archive/master.tar.gz
tar xzf cot.tgz
cd cot-master
```

Install the COT libraries and script

```
sudo python setup.py install
```

Installing helper programs

Certain COT features require helper programs - you can install these as part of the COT installation process, or they can be installed as-needed by COT:

- COT uses `qemu-img` as a helper program for various operations involving the creation, inspection, and modification of hard disk image files packaged in an OVF.
- The `cot add-disk` command requires either `qemu-img` (version 2.1 or later) or `vmdktool` as a helper program when adding hard disks to an OVF.
- The `cot inject-config` command requires `mkisofs` (or its fork `genisoimage`) and/or `xorriso` to create ISO (CD-ROM) images for platforms that use ISOs to package the configuration.

- Similarly, for platforms using hard disks for bootstrap configuration, `cot inject-config` requires `fatdisk` to format hard disk images.
- The `cot deploy ... esxi` command requires `ovftool` to communicate with an ESXi server. If `ovftool` is installed, COT's automated unit tests will also make use of `ovftool` to perform additional verification that OVFs and OVAs created by COT align with VMware's expectations for these file types.

COT can attempt to install these tools using the appropriate package manager for your platform (i.e., `MacPorts` for Mac OS X, and either `apt-get` or `yum` for Linux).

Warning: Unfortunately, VMware requires a site login to download `ovftool`, so if you need this tool, you will have to install it yourself. COT cannot install it for you at present.

To let COT attempt to pre-install all of the above helpers, you can optionally run:

```
cot install-helpers
```

See [here](#) for more details.

If you skip this step, then when you are running COT, and it encounters the need for a helper that has not been installed, COT will prompt you to allow it to install the helper in question.

Using COT

Getting CLI help

Synopsis

```
cot --help
cot --version
cot help <command>
cot <command> --help
cot <options> <command> <command-options>
```

Description

Common OVF Tool (COT), version 1.9.1+0.g6da4345.dirty

A tool for editing Open Virtualization Format (.ovf, .ova) virtual appliances, with a focus on virtualized network appliances such as the Cisco CSR 1000V and Cisco IOS XRv platforms.

You can always get detailed help for COT by running `cot --help`, `cot <command> --help`, or `cot help <command>`.

Options

-h, --help	show this help message and exit
-V, --version	show program's version number and exit
-f, --force	Perform requested actions without prompting for confirmation
-q, --quiet	Quiet output and logging (warnings and errors only)
-v, --verbose	Verbose output and logging
-d, --debug	Debug (most verbose) output and logging

Commands

add-disk Add a disk image to an OVF package and map it as a disk in the guest environment

add-file Add a file to an OVF package

deploy Create a new VM on the target hypervisor from the given OVF or OVA

edit-hardware Edit virtual machine hardware properties of an OVF

edit-product Edit product info in an OVF

edit-properties Edit or create environment properties of an OVF

help Print help for a command

info Generate a description of an OVF package

inject-config Inject a configuration file into an OVF package

install-helpers Install/verify COT manual pages and any third- party helper programs that COT may require

remove-file Remove a file from an OVF package

Verifying and installing helper programs with `cot install-helpers`

Synopsis

```
cot install-helpers --help
cot <opts> install-helpers --verify-only
cot <opts> install-helpers [--ignore-errors]
```

Description

Install or verify the installation of COT manual pages and various required third-party helper programs for COT.

- `qemu-img` (<http://www.qemu.org/>)
- `mkisofs` (<http://cdrecord.org/>)
- `ovftool` (<https://www.vmware.com/support/developer/ovf/>)
- `fatdisk` (<http://github.com/goblinhack/fatdisk>)
- `vmdktool` (<http://www.freshports.org/sysutils/vmdktool/>)

Options

- | | |
|----------------------------|--|
| -h, --help | show this help message and exit |
| --verify-only | Only verify helpers – do not attempt to install any missing helpers. |
| -i, --ignore-errors | Do not fail even if helper installation fails. |

Examples

Verify whether COT can find all expected helper programs

```
> cot install-helpers --verify-only
Results:
-----
COT manpages: present in /usr/share/man/man1/
fatdisk:      present at /opt/local/bin/fatdisk
mkisofs:      present at /opt/local/bin/mkisofs
ovftool:      present at /usr/local/bin/ovftool
qemu-img:     present at /opt/local/bin/qemu-img
vmdktool:     NOT FOUND
```

Have COT attempt to install missing helpers for you. Note that most helpers require administrator / sudo privileges to install. If any installation fails, COT will exit with an error, unless you pass `--ignore-errors`.

```
> cot install-helpers
  INFO: Installing 'fatdisk'...
  INFO: Compiling 'fatdisk'
  INFO: Calling './RUNME'...
(...)
  INFO: ...done
  INFO: Compilation complete, installing to /usr/local/bin
  INFO: Successfully installed 'fatdisk'
  INFO: Calling 'fatdisk --version' and capturing its output...
  INFO: ...done
  INFO: Installing 'vmdktool'...
  INFO: vmdktool requires 'zlib'... installing 'zlib'
  INFO: Calling 'dpkg -s zlib1g-dev' and capturing its output...
  INFO: ...done
  INFO: Compiling 'vmdktool'
  INFO: Calling 'make CFLAGS="-D_GNU_SOURCE -g -O -pipe"'...
(...)
  INFO: ...done
  INFO: Compilation complete, installing to /usr/local
  INFO: Calling 'make install'...
install -s vmdktool /usr/local/bin/
install vmdktool.8 /usr/local/man/man8/
  INFO: ...done
  INFO: Successfully installed 'vmdktool'
  INFO: Calling 'vmdktool -V' and capturing its output...
  INFO: ...done
  INFO: Copying cot-add-disk.1 to /usr/share/man/man1/cot-add-disk.1
(...)
  INFO: Copying cot.1 to /usr/share/man/man1/cot.1
Results:
-----
COT manpages: successfully installed to /usr/share/man
fatdisk:      successfully installed to /usr/local/bin/fatdisk
mkisofs:      present at /usr/bin/mkisofs
ovftool:      INSTALLATION FAILED: No support for automated
              installation of ovftool, as VMware requires a site
              login to download it. See
              https://www.vmware.com/support/developer/ovf/
qemu-img:     present at /usr/bin/qemu-img
vmdktool:     successfully installed to /usr/local/bin/vmdktool

Unable to install some helpers
```

Warning: Unfortunately, VMware requires a site login to download `ovftool`, so if you need this tool, you will have to install it yourself. COT cannot install it for you at present.

Inspecting OVF contents with `cot info`

Synopsis

```
cot info --help
cot info [-b | -v] PACKAGE [PACKAGE ...]
```

Description

Show a summary of the contents of the given OVF(s) and/or OVA(s).

Options

PACKAGE [**PACKAGE ...**] OVF descriptor(s) and/or OVA file(s) to describe

- h, --help** show this help message and exit
- b, --brief** Brief output (shorter)
- v, --verbose** Verbose output (longer)

Updating product information with `cot edit-product`

Synopsis

```
cot edit-product --help
cot <opts> edit-product PACKAGE [-o OUTPUT] [-c PRODUCT_CLASS]
                                [-p PRODUCT] [-n VENDOR] [-v SHORT_VERSION]
                                [-V FULL_VERSION] [-u PRODUCT_URL ]
                                [-r VENDOR_URL] [-l APPLICATION_URL]
```

Description

Edit product information attributes of the given OVF or OVA

Options

PACKAGE OVF descriptor or OVA file to edit

- h, --help** show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new OVF/OVA package to create instead of updating the existing OVF
- c PRODUCT_CLASS, --product-class PRODUCT_CLASS** Product class, such as “com.cisco.csr1000v”
- p PRODUCT, --product PRODUCT** Product name string, such as “Cisco IOS-XE”

- n VENDOR, --vendor VENDOR** Vendor string, such as “Cisco Systems, Inc.”
- v SHORT_VERSION, --version SHORT_VERSION** Software short version string, such as “15.3(4)S” or “5.2.0.01I”
- V FULL_VERSION, --full-version FULL_VERSION** Software long version string, such as “Cisco IOS-XE Software, Version 15.3(4)S”
- u PRODUCT_URL, --product-url PRODUCT_URL** Product URL, such as “<http://www.cisco.com/go/iosxrv>”
- r VENDOR_URL, --vendor-url VENDOR_URL** Vendor URL, such as “<http://www.cisco.com>”
- l APPLICATION_URL, --application-url APPLICATION_URL** Application URL, such as “<https://router1:530/>”

Adding disks to an OVF with `cot add-disk`

Synopsis

```
cot add-disk --help
cot <opts> add-disk DISK_IMAGE PACKAGE [-o OUTPUT] [-f FILE_ID]
                        [-t {harddisk,cdrom}] [-c {ide,scsi}]
                        [-s SUBTYPE] [-a ADDRESS] [-d DESCRIPTION]
                        [-n DISKNAME]
```

Description

Add or replace a disk image in the specified OVF or OVA. If the specified disk image, controller/address, file-id, and/or instance match an existing entry in the OVF, will replace the existing disk with the provided file (prompting for confirmation if `-force` was not set); otherwise, will create a new disk entry.

Options

DISK_IMAGE Disk image file to add to the package

PACKAGE OVF descriptor or OVA file to edit

General options

- h, --help** Show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new OVF/OVA package to create instead of updating the existing OVF

Disk-related options

- f FILE_ID, --file-id FILE_ID** Disk image file ID string within the OVF package (default: use disk image filename)
- t <harddisk,cdrom>, --type <harddisk,cdrom>** Disk drive type (default: files ending in `.vmdk/.raw/.qcow2/.img` will use `harddisk` and files ending in `.iso` will use `cdrom`)

Controller-related options

- c <ide,scsi>, --controller <ide,scsi>** Disk controller type (default: determined by disk drive type and platform)
- a ADDRESS, --address ADDRESS** Address of the disk, such as “1:0”. Requires that `--controller` be explicitly set. (default: use first unused address on the controller)
- s SUBTYPE, --subtype SUBTYPE** Disk controller subtype such as “virtio” or “lsilogic”.

Descriptive options

- d DESCRIPTION, --description DESCRIPTION** Description of this disk (optional)
- n DISKNAME, --name DISKNAME** Name of this disk (default: “Hard disk #” or “CD-ROM #” as appropriate)

Packaging additional files into an OVF with `cot add-file`

Synopsis

```
cot add-file --help
cot <opts> add-file FILE PACKAGE [-o OUTPUT] [-f FILE_ID]
```

Description

Add or replace a file in the given OVF. If the specified file and/or file-id match existing package contents, will replace it (prompting for confirmation if `--force` was not set); otherwise, will create a new file entry.

Options

FILE File to add to the package

PACKAGE Package, OVF descriptor or OVA file to edit

- h, --help** show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new VM package to create instead of updating the existing package
- f FILE_ID, --file-id FILE_ID** File ID string within the package (default: same as filename)

Removing files from an OVF with `cot remove-file`

Synopsis

```
cot remove-file --help
cot <opts> remove-file [-f FILE_PATH] [-i FILE_ID] PACKAGE
                        [-o OUTPUT]
```

Description

Remove a file from the given OVF. Will prompt for confirmation unless `--force` is set.

Options

PACKAGE Package, OVF descriptor or OVA file to edit

General options

- h, --help** Show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new OVF/OVA package to create instead of updating the existing OVF

File selection options

- f FILE_PATH, --file-path FILE_PATH** File name or path within the package
- i FILE_ID, --file-id FILE_ID** File ID string within the package

Customizing hardware profiles with `cot edit-hardware`

Synopsis

```
cot edit-hardware --help
cot <opts> edit-hardware PACKAGE [-o OUTPUT] -v TYPE [TYPE2 ...]
cot <opts> edit-hardware PACKAGE [-o OUTPUT]
                                [-p PROFILE [PROFILE2 ...]
                                [--delete-all-other-profiles]] [-c CPUS]
                                [-m MEMORY] [-n NICS]
                                [--nic-types TYPE [TYPE2 ...]]
                                [-N NETWORK [NETWORK2 ...]]
                                [-M MAC1 [MAC2 ...]]
                                [--nic-names NAME1 [NAME2 ...]]
                                [-s SERIAL_PORTS] [-S URI1 [URI2 ...]]
                                [--scsi-subtypes TYPE [TYPE2 ...]]
                                [--ide-subtypes TYPE [TYPE2 ...]]
```

Description

Edit hardware properties of the specified OVF or OVA

Options

PACKAGE OVF descriptor or OVA file to edit

General options

- h, --help** Show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new OVF/OVA package to create instead of updating the existing OVF
- v <TYPE...>, --virtual-system-type <TYPE...>** Change virtual system type(s) supported by this OVF/OVA package.
- p <PROFILE...>, --profiles <PROFILE...>** Make hardware changes only under the given configuration profile(s). (default: changes apply to all profiles)
- delete-all-other-profiles** Delete all configuration profiles other than those specified with the `-p` option

Computational hardware options

- c CPUS, --cpus CPUS** Set the number of CPUs.
- m MEMORY, --memory MEMORY** Set the amount of RAM. (Examples: “4096M”, “4 GiB”)

Network interface options

- n NICS, --nics NICS** Set the number of NICs.
- nic-types <TYPE...>** Set the hardware type(s) for all NICs. (default: do not change existing NICs, and new NICs added will match the existing type(s).)
- nic-names <NAME1...>** Specify a list of one or more NIC names or patterns to apply to NIC devices. See Notes.
- N <NETWORK...>, --nic-networks <NETWORK...>** Specify a series of one or more network names or patterns to map NICs to. See Notes.
- network-descriptions <NAME1...>** Specify a list of one or more network descriptions or patterns to apply to the networks. See Notes.
- M <MAC1...>, --mac-addresses-list <MAC1...>** Specify a list of MAC addresses for the NICs. If N MACs are specified, the first (N-1) NICs will receive the first (N-1) MACs, and all remaining NICs will receive the Nth MAC

Serial port options

- s SERIAL_PORTS, --serial-ports SERIAL_PORTS** Set the number of serial ports.
- S <URI1...>, --serial-connectivity <URI1...>** Specify a series of connectivity strings (URIs such as “telnet://localhost:9101”) to map serial ports to. If fewer URIs than serial ports are specified, the remaining ports will be unmapped.

Disk and disk controller options

- scsi-subtypes <TYPE...>** Set resource subtype(s) (such as “lsilogic” or “virtio”) for all SCSI controllers. If an empty string is provided, any existing subtype will be removed.
- ide-subtypes <TYPE...>** Set resource subtype(s) (such as “virtio”) for all IDE controllers. If an empty string is provided, any existing subtype will be removed.

Notes

The `--nic-names`, `--nic-networks`, and `--network-descriptions` options support the use of a wildcard value to automatically generate a series of consecutively numbered strings. The syntax for the wildcard option is `'{'` followed by a number to start incrementing from, followed by `'}'`. See examples below.

Examples

Create a new profile named “1CPU-8GB” with 1 CPU and 8 gigabytes of RAM

```
cot edit-hardware csr1000v.ova --output csr1000v_custom.ova \
  --profile 1CPU-4GB --cpus 1 --memory 8GB
```

Wildcard example - without caring about how many NICs are defined in the input OVA, rename all of the NICs in the output OVA as ‘Ethernet0/10’, ‘Ethernet0/11’, ‘Ethernet0/12’, etc., and map them to networks ‘Ethernet0_10’, ‘Ethernet0_11’, ‘Ethernet0_12’, etc., which are described as ‘Data network 1’, ‘Data network 2’, etc.

```
cot edit-hardware input.ova -o output.ova \
  --nic-names "Ethernet0/{10}" \
  --nic-networks "Ethernet0_{10}" \
  --network-descriptions "Data network {1}"
```

Combination of fixed and wildcarded names - rename the NICs in the output OVA as ‘mgmt’, ‘eth0’, ‘eth1’, ‘eth2’...

```
cot edit-hardware input.ova -o output.ova --nic-names "mgmt" \
  "eth{0}"
```

Customizing OVF environment settings with `cot edit-properties`

Synopsis

```
cot edit-properties --help
cot <opts> edit-properties PACKAGE
                                [-p KEY1=VALUE1 [-p KEY2=VALUE2 ...]]
                                [-l LABEL1 [-l LABEL2 ...]]
                                [-d DESC1 [-d DESC2 ...]] [-c CONFIG_FILE]
                                [-u [USER_CONFIGURABLE]]
                                [-t TRANSPORT [TRANSPORT2 ...]]
                                [-o OUTPUT]
cot <opts> edit-properties PACKAGE [-u [USER_CONFIGURABLE]]
                                [-o OUTPUT]
```

Description

Configure environment properties of the given OVF or OVA. The user may specify keys and values as command-line arguments or may provide a config-file to read from. If neither `--config-file`, `--properties`, nor `--transport` are given, the program will run interactively.

Options

PACKAGE OVF descriptor or OVA file to edit

General options

- h, --help** Show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new OVF/OVA package to create instead of updating the existing OVF

Property setting options

- u <USER_CONFIGURABLE>, --user-configurable <USER_CONFIGURABLE>** Update the 'userConfigurable' flag on all edited properties to True or the given value
- c CONFIG_FILE, --config-file CONFIG_FILE** Read configuration CLI from this text file and generate generic properties for each line of CLI
- p <KEY1[=VALUE1][+TYPE1]...>, --properties <KEY1[=VALUE1][+TYPE1]...>** Update or create the given property keys. A '=' delimits the optional value to set this key to. A '+' delimits the optional type to enforce for this key. This argument may be repeated as needed to specify multiple properties to edit.
- l <LABEL1...>, --labels <LABEL1...>** Set the label(s) for the property(s) being edited. If this option is specified, the number of properties and the number of labels *must* be equal.
- d <DESC1...>, --descriptions <DESC1...>** Set the description(s) for the property(s) being edited. If this option is specified, the number of properties and the number of descriptions *must* be equal.
- t <TRANSPORT...>, --transports <TRANSPORT...>** Set the transport method(s) for properties. Known values are 'iso', 'vmware', and 'ibm', or an arbitrary URI may be specified.

Examples

Add configuration from a text file and mark the resulting properties as non-user-configurable.

```
cot edit-properties input.ovf -c config.txt -u=0
```

Add/update two properties, one a string with no default value and the other a boolean defaulting to true, and mark both properties as user-configurable.

```
cot edit-properties input.ovf -p string-property+string \  
-p bool-property=true+boolean --user-configurable
```

Update the label and description of two existing properties

```
cot edit-properties input.ovf -p hostname -l "Hostname" \  
-d "Hostname of this device" -p enable-ssh -l "Enable \  
remote SSH access" -d "Enable sshd and disable telnetd"
```

Embedding bootstrap configuration with `cot inject-config`

Synopsis

```
cot inject-config --help
cot <opts> inject-config PACKAGE [-o OUTPUT] [-c CONFIG_FILE]
                                [-s SECONDARY_CONFIG_FILE]
                                [-e EXTRA_FILE [EXTRA_FILE2 ...]]
```

Description

Add one or more “bootstrap” configuration file(s) to the given OVF or OVA. These files will be packaged into a virtual hard disk, or virtual CD-ROM, as appropriate to the target platform. Any specified primary and secondary config files will be renamed if necessary to meet expectations of the target platform, while any files provided with the `--extra-files` option will be included as-is and will not be renamed.

Options

PACKAGE Package, OVF descriptor or OVA file to edit

- h, --help** show this help message and exit
- o OUTPUT, --output OUTPUT** Name/path of new VM package to create instead of updating the existing package
- c CONFIG_FILE, --config-file CONFIG_FILE** Text file to embed as primary configuration
- s SECONDARY_CONFIG_FILE, --secondary-config-file SECONDARY_CONFIG_FILE** Text file to embed as secondary configuration (currently only used for IOS XR admin config)
- e <EXTRA_FILE...>, --extra-files <EXTRA_FILE...>** Additional file(s) to include as-is

Deploying an OVF to create a VM with `cot deploy`

Synopsis

```
cot deploy --help
cot <opts> deploy PACKAGE esxi ...
```

Description

Deploy an OVF or OVA to create a virtual machine on a specified server.

Options

PACKAGE OVF descriptor or OVA file

- h, --help** show this help message and exit

Hypervisors

esxi Deploy to ESXi, vSphere, or vCenter

Creating a VM on VMware vCenter/vSphere with `cot deploy esxi`

Synopsis

```
cot deploy PACKAGE esxi --help
cot <opts> deploy PACKAGE esxi LOCATOR [-u USERNAME] [-p PASSWORD]
                                     [-c CONFIGURATION] [-n VM_NAME] [-P]
                                     [-N OVF1=HOST1 [-N OVF2=HOST2 ...]]
                                     [-S KIND1:VAL1[,OPTS1]
                                     [-S KIND2:VAL2[,OPTS2] ...]]
                                     [-d DATASTORE] [-o=OVFTOOL_ARGS]
```

Description

Deploy OVF/OVA to ESXi/vCenter/vSphere hypervisor

Options

LOCATOR vSphere target locator. Examples: “192.0.2.100” (deploy directly to ESXi server), “192.0.2.101/mydatacenter/host/192.0.2.100” (deploy via vCenter server)

- h, --help** show this help message and exit
- u USERNAME, --username USERNAME** Server login username
- p PASSWORD, --password PASSWORD** Server login password
- c CONFIGURATION, --configuration CONFIGURATION** Use the specified configuration profile defined in the OVF. If unspecified and the OVF has multiple profiles, the user will be prompted or the default configuration will be used.
- n VM_NAME, --vm-name VM_NAME** Name to use for the VM (if applicable) and any files created. If unspecified, the name of the OVF will be used.
- P, --power-on** Power on the created VM to begin booting immediately.
- N <OVF_NET1=HOST_NET1...>, --network-map <OVF_NET1=HOST_NET1...>** Map networks named in the OVF to networks (bridges, vSwitches, etc.) in the hypervisor environment. This argument may be repeated as needed to specify multiple mappings.
- S <K1:V1[,O1]...>, --serial-connection <K1:V1[,O1]...>** Set connectivity for a serial port defined in the OVF. This argument may be repeated to specify more port connections. Each entry should be structured as ‘kind:value’ or ‘kind:value,options’.
- d DATASTORE, --datastore DATASTORE** ESXi datastore to use for the new VM
- o OVFTOOL_ARGS, --ovftool-args OVFTOOL_ARGS** Quoted string describing additional CLI parameters to pass through to “ovftool”. Examples: `-o="--foo"`, `--ovftool-args="--foo --bar"`

Examples

Deploy to vSphere/ESXi server 192.0.2.100 with credentials admin/admin, creating a VM named ‘test_vm’ from foo.ova.

```
cot deploy foo.ova esxi 192.0.2.100 -u admin -p admin \  
-n test_vm
```

Deploy to vSphere/ESXi server 192.0.2.100, with username admin (prompting the user to input a password at runtime), creating a VM based on profile '1CPU-2.5GB' in foo.ova, and creating the serial port as a telnet server listening on port 10022 of the host

```
cot deploy foo.ova esxi 192.0.2.100 -u admin -c 1CPU-2.5GB \  
-S telnet://:10022,server
```

Deploy to vSphere server 192.0.2.1 which belongs to datacenter 'mydc' on vCenter server 192.0.2.100, and map the two NIC networks to vSwitches. Note that in this case -u specifies the vCenter login username.

```
cot deploy foo.ova esxi "192.0.2.100/mydc/host/192.0.2.1" \  
-u administrator -N "GigabitEthernet1=VM Network" \  
-N "GigabitEthernet2=myvswitch"
```

Deploy with passthrough arguments to ovftool.

```
cot deploy foo.ova esxi 192.0.2.100 -u admin -p password \  
--ovftool-args="--overwrite --acceptAllEulas"
```

Glossary

COT Common *OVF* Tool

controller, hardware controller A virtual hardware controller for hardware such as a *disk device*. In addition to its primary type (IDE, SCSI, etc.), a controller may also have a subtype, such as `virtio` or `lsilogic`. In an *OVF* package, a controller is represented by an XML `Item` element in the `VirtualHardwareSection` of the *OVF descriptor*. Typically each *disk device* must be associated with a controller.

disk description, disk element, disk reference A description of a virtual disk included in a virtual machine. In an *OVF descriptor*, this is an XML `Disk` element in the `DiskSection`. This disk description may be associated with a *file reference* and/or *disk file*, or it may be a placeholder for a blank disk not yet created. Typically a disk description must be associated with a *disk drive* in order to actually be accessible by the guest OS.

disk device, disk drive, disk item A *hardware item* describing a virtual CD-ROM, DVD-ROM, or hard disk drive. In an *OVF* package, this is an XML `Item` element in the `VirtualHardwareSection` of the *OVF descriptor*. This item may reference a *disk reference* or a *file reference* to map a filesystem to this drive. Typically a disk drive must be associated to a *hardware controller*.

disk file, disk image A file such as a `.vmdk`, `.iso`, or `.qcow2`. May or may not be associated with a *disk drive*.

file element, file reference A reference to a file, such as a *disk file* or any other file type, to be included in a virtual machine. In an *OVF descriptor*, this is an XML `File` element in the `References` section.

hardware element, hardware item Generic term for any discrete piece of virtual machine hardware, including but not limited to the CPU(s), memory, *disk drive*, *hardware controller*, network port, etc.

OVF Open Virtualization Format, an open standard.

OVF descriptor An XML file, based on the *OVF* specification, which describes a virtual machine.

Change Log

All notable changes to the COT project will be documented in this file. This project adheres to [Semantic Versioning](#).

1.9.1 - 2017-02-21

Changed

- Removed 32 MiB memory limitation on Cisco IOS XRv 9000 platform.

1.9.0 - 2017-02-13

Added

- Support for Python 3.6
- Support for *brew* package manager ([#55](#)).
- Support for Cisco Nexus 9000v (NX-OSv 9000) platform ([#60](#)).

Fixed

- Improved messaging when COT is unable to install a helper program ([#57](#)).

1.8.2 - 2017-01-18

Fixed

- Issue ([#58](#)) where various multi-value CLI options for the `edit-hardware` and `inject-config` commands did not append properly.
- Issue in which explicitly specified NIC names were being overwritten by names auto-derived from network names when attempting to set both NIC name and network names in a single `cot edit-hardware` call.
- `cot edit-properties` again accepts property values containing the characters `+` and `=` ([#59](#)).

Added

- COT can now detect the presence of an `.ovf` file in a TAR archive even if the archive is not named `*.ova*` and even if the `.ovf` file is not the first file in the archive as specified by the OVF specification. This allows (read-only at present) handling of VirtualBox Vagrant `.box` files, which are approximately equivalent to non-standards-compliant OVAs.

1.8.1 - 2016-11-12

Fixed

- Under Python versions older than 2.7.9, explicitly require `pyOpenSSL` and `ndg-httpsclient` to avoid issues like `hostname 'people.freebsd.org'` doesn't match `'wfe0.ysv.freebsd.org'` when installing `vmtool`.

1.8.0 - 2016-11-08

Fixed

- `TypeError` in `find_item` method (#54).
- `cot inject-config` correctly handles OVAs with multiple empty CD-ROM drives to choose amongst (#54 also).
- Cisco CSR1000v platform now supports 8 CPUs as a valid option.

Added

- `cot inject-config --extra-files` parameter (#53).
- Helper class for `isoinfo` (a companion to `mkisofs`).
- Added glossary of terms to COT documentation.
- Inline documentation (docstrings) are now validated using the [Pylint docparams](#) extension.

Changed

- Refactored the monolithic `COT/platforms.py` file into a proper submodule.
- `create_iso()` now adds Rock Ridge extensions by default.
- Refactored `COT.helpers` into two modules - `COT.helpers` (now just for handling helper programs such as `apt-get` and `mkisofs`) and `COT.disks` (which uses the helpers to handle ISO/VMDK/QCOW2/RAW image files).
- Inline documentation (docstrings) have been converted to “Google style” for better readability in the code. Sphinx rendering of documentation (for [readthedocs.org](#), etc) now makes use of the [napoleon](#) extension to handle this style.

Removed

- `get_checksum()` is no longer part of the `COT.helpers` API. (It's now the method `file_checksum()` in `COT.data_validation`, where it really belonged from the start).
- `download_and_expand()` is no longer part of the `COT.helpers` public API. (It's now the static method `download_and_expand_tgz()` on class `Helper`.)

1.7.4 - 2016-09-21

Newer versions of Sphinx have dropped support for Python 2.6 and 3.3, so I have updated COT's installation requirements to use older Sphinx versions under these Python versions.

1.7.3 - 2016-09-06

Added

- When adding NICs to an OVF, if no `-nic-networks` are specified, `cot edit-hardware` will now try to infer sequential naming of the Network elements and if successful, offer to create additional Networks as appropriate. (#18)

1.7.2 - 2016-08-17

Fixed

- Issue #52 - OVFItemDataError raised when adding NICs to CSR1000V OVA, or more generally when cloning an OVFItem whose ElementName references its Connection.

1.7.1 - 2016-08-12

Fixed

- `cot deploy ... --serial-connection` will create additional serial ports beyond those defined in the OVF, if requested. Previously it would ask the user for confirmation but not actually do anything about it. (#51)

1.7.0 - 2016-08-05

Added

- Support for Python 3.5
- Enhancements to `cot edit-properties` (#50):
 - Added `--user-configurable` option to set whether created/updated properties are marked as user-configurable in the OVF.
 - Added `--labels` and `--descriptions` options to set/update the labels and descriptions associated with properties.
 - It's now valid to set no default value for a property by omitting the `=value`, as in `-p property-with-no-value`, as well as the existing `-p property-with-empty-value=` syntax to set an empty string as the value.
 - Users can now optionally specify the property type to enforce for each property by using the delimiter `+type`, as in `-p key=1+boolean`.

Changed

- Switched from statement coverage to branch coverage for better test analysis.
- Moved from [Coveralls](#) to [Codecov](#) for test coverage tracking, since Coveralls does not support branch coverage reporting.

Fixed

- When `cot edit-hardware` is used to create new NICs in an OVF that previously had none, and the user does not specify any corresponding Network entries, automatically create a ‘VM Network’ entry, because all NICs must be mapped to Networks for a valid OVF descriptor.

1.6.1 - 2016-07-07

Fixed

- `ValueMismatchError` exceptions are properly caught by the CLI wrapper so as to result in a graceful exit rather than a stack trace.
- `cot remove-file` now errors if the user specifies both `file-id` and `file-path`, one of which matches a file in the OVF, but the other does not match this or any other file.
- Better handling of exceptions and usage of `sudo` when installing helpers.
- Manual pages are now correctly included in the distribution. Oops!

1.6.0 - 2016-06-30

Added

- `cot edit-product --product-class` option, to set or change the product class identifier (such as `com.cisco.csr1000v`).
- Enabled additional code quality validation with `Pylint`, `pep8-naming`, and `mccabe` (#49).

Changed

- Lots of refactoring to reduce code complexity as measured by `Pylint` and `mccabe`.

Fixed

- COT now recognizes `AllocationUnits` values like megabytes.
- COT no longer ignores the `AllocationUnits` value given for RAM.
- `COT.ovf.byte_string()` now properly uses binary units (KiB rather than kB, etc.)

1.5.2 - 2016-06-17

Changed

- Development requirement changes: The package `pep8` has been renamed to `pycodestyle`, and `pep257` has been renamed to `pydocstyle`. Updated configuration and documentation to reflect these changes. Also, `flake8-pep257` does not presently handle these changes, so replaced it as a dependency with the more up-to-date `flake8-docstrings` package.

1.5.1 - 2016-06-07

Added

- `cot edit-hardware --network-descriptions` option, to specify the descriptive string(s) associated with each network definition.

Fixed

- [#48](#) - NIC type not set when adding NICs to an OVF that had none before.
- When updating NIC network mapping, COT now also updates any Description that references the network mapping.

1.5.0 - 2016-06-06

Added

- [#47](#) - Added `cot remove-file` subcommand.
- [#43](#) - add `cot edit-properties --transport` option to set environment transport type(s) - iso, VMWare Tools, etc.
 - `cot info` now has a new “Environment” section that displays the transport type
- [#45](#) - support for multiple values for `--nic-types`, `--ide-subtypes`, and `--scsi-subtypes` in `cot edit-hardware`.
- COT now recognizes the Cisco IOS XRv 9000 platform identifier `com.cisco.ios-xrv9000`.
- [#21](#) - subcommand aliases (Python 3.x only):
 - `cot edit-product` aliases: `cot set-product`, `cot set-version`
 - `cot edit-properties` aliases: `cot set-properties`, `cot edit-environment`, `cot set-environment`
 - `cot info` alias: `cot describe`
 - `cot inject-config` alias: `cot add-bootstrap`
 - `cot remove-file` alias: `cot delete-file`
- Support for tab-completion of CLI parameters using [argcomplete](#).

Changed

- `cot edit-hardware` options `--nic-types`, `--ide-subtypes`, and `--scsi-subtypes` are now validated and canonicalized by COT, meaning that:
 - `cot edit-hardware --nic-type virtio-net-pci` is now a valid command and will correctly create an OVF with `ResourceSubType virtio` (not `virtio-net-pci`)
 - `cot edit-hardware --ide-subtype foobar` will now fail with an error
- `cot info` is now more self-consistent in how it displays property keys. They are now always wrapped in `<>`, whereas previously this was only sometimes the case.
- `cot info --verbose` now displays file and disk ID strings under the “Files and Disks” section.

1.4.2 - 2016-05-11

Added

- COT now supports `xorriso` as another alternative to `mkisofs` and `genisoimage`

Fixed

- [#42](#) - `cot deploy esxi` error handling behavior needed to be updated for [requests](#) release 2.8.

- [#44](#) - test case failure seen when running `pyVmomi` 6.0.0.2016.4.

Changed

- Installation document now recommends installation via `pip` rather than installing from source.
- [#40](#) - Now uses faster Docker-based infrastructure from [Travis CI](#) for CI builds/tests.

1.4.1 - 2015-09-02

Fixed

- [#41](#) - symlinks were not dereferenced when writing out to OVA.

1.4.0 - 2015-09-01

Added

- [#24](#) - `cot deploy esxi` now creates serial ports after deployment using `pyVmomi` library.
 - Serial port connectivity must be specified either via entries in the OVF (which can be defined using `cot edit-hardware ... -S`) or at deployment time using the new `-S / --serial-connection` parameter to `cot deploy`.
 - The syntax for serial port connectivity definition is based on that of QEMU's `--serial` CLI option.
 - Currently only “telnet”, “tcp”, and “device” connection types are supported.
- [#38](#) - `cot edit-product` can now set product and vendor information.
- `flake8` validation now includes `pep257` to validate docstring compliance to [PEP 257](#) as well.
- Added changelog file.
- Added `COT.file_reference` submodule in support of [#39](#).

Changed

- Split ESXi-specific logic out of `COT.deploy` module and into new `COT.deploy_esxi` module.
- UT for `COT.deploy_esxi` now requires `mock` (standard library in Python 3.x, install via `pip` on Python 2.x).

Fixed

- [#39](#) - avoid unnecessary file copies to save time and disk space.

1.3.3 - 2015-07-02

Fixed

- [#10](#) - When changing network mapping, delete no longer needed networks
- [#31](#) - Added `--delete-all-other-profiles` option to `cot edit-hardware`
- [#32](#) - `cot edit-hardware` network names can now use wildcards
- [#34](#) - `cot add-disk` can now be used to replace a CD-ROM drive with a hard disk, or vice versa.

1.3.2 - 2015-04-09

Fixed

- Adapt to changes to the Travis-CI testing environment.

1.3.1 - 2015-04-09

Fixed

- [#30](#) - `cot install-helpers` can now install `fatdisk` and `vmdktool` under Python 3.

1.3.0 - 2015-03-27

Added

- Installation of helper programs is now provided by a `cot install-helpers` subcommand rather than a separate script.
- COT now has man pages (`man cot`, `man cot-edit-hardware`, etc.) The man pages are also installed by `cot install-helpers`.
- Improved documentation of the CLI on readthedocs.org as well.

Changed

- Refactored `COT.helper_tools` module into `COT.helpers` subpackage. This package has an API (`COT.helpers.api`) for the rest of COT to access it; the helper-specific logic (`qemu-img`, `fatdisk`, etc.) is split into individual helper modules that are abstracted away by the API.
- Similarly, logic from `COT.tests.helper_tools` has been refactored and enhanced under `COT.helpers.tests`.
- Renamed all test code files from “`foo.py`” to “`test_foo.py`” to facilitate test case discovery.
- CLI help strings are dynamically rendered to ReST when docs are built, providing cleaner output for both readthedocs.org and the manpages.

Removed

- COT no longer supports Python 3.2.
- `cot_unittest` is no more - use `tox` or `unit2 discover` to run tests.
- As noted above, the installation script `check_and_install_helpers.py` no longer exists - this functionality is now provided by the `COT.install_helpers` module.

1.2.4 - 2015-03-06

Fixed

- [#29](#) - `cot edit-properties` interactive mode was broken in v1.2.2

1.2.3 - 2015-02-19

Fixed

- Some documentation fixes for <http://cot.readthedocs.org>

1.2.2 - 2015-02-19

Added

- Documentation built with Sphinx and available at <http://cot.readthedocs.org>

Changed

- CLI adapts more intelligently to terminal width (fixes #28)
- Submodules now use Python properties instead of `get_value/set_value` methods.

1.2.1 - 2015-02-03

Added

- Now PEP 8 compliant - passes validation by [flake8](#) code analysis.
- Very preliminary support for OVF 2.x format
- Now uses [tox](#) for easier test execution and [coverage.py](#) for code coverage analysis.
- Code coverage reporting with [Coveralls](#).

Changed

- Now uses [colorlog](#) instead of `coloredlogs` for CLI log colorization, as this fits better with COT's logging model.
- Greatly improved unit test structure and code coverage, including tests for logging.

1.2.0 - 2015-01-16

Added

- Greatly improved logging (#26). COT now defaults to logging level INFO, which provides relatively brief status updates to the user. You can also run with `--quiet` to suppress INFO messages and only log WARNING and ERROR messages, `--verbose` to see VERBOSE messages as well, or `--debug` if you want to really get into the guts of what COT is doing.
- Now integrated with [Travis CI](#) for automated builds and UT under all supported Python versions. This should greatly improve the stability of COT under less-common Python versions. (#12)

Changed

- The CLI for `cot deploy` has been revised somewhat based on user feedback.
- A lot of restructuring of the underlying code to make things more modular and easier to test in isolation.

Fixed

- Various bugfixes for issues specific to Python 2.6 and 3.x - these environments should now be fully working again.

1.1.6 - 2015-01-05

Added

- Added THANKS file recognizing various non-code contributions to COT.

Fixed

- Bug fixes for `cot inject-config` and `cot deploy`, including issues [#19](#) and [#20](#) and a warning to users about serial ports and ESXi (issue eventually to be addressed by fixing [#24](#)).
- More graceful handling of Ctrl-C interrupt while COT is running.

1.1.5 - 2014-11-25

Fixed

- Fixed issue [#17](#) (`cot edit-hardware` adding NICs makes an OVA that vCenter regards as invalid)
- Removed several spurious WARNING messages

1.1.4 - 2014-11-12

Added

- COT can at least be installed and run under CentOS/Python2.6 now, although the automated unit tests will complain about the different XML output that 2.6 produces.

Changed

- Vastly improved installation workflow under Linuxes supporting `apt-get` or `yum` - included helper script can automatically install all helper programs except `ovftool`. Fixes [#9](#).

Fixed

- Improved `cot deploy` handling of config profiles - fixed [#5](#) and [#15](#)

1.1.3 - 2014-10-01

Added

- `cot edit-hardware` added `--nic-names` option for assigning names to each NIC
- `cot info` now displays NIC names.

Fixed

- Improved installation documentation
- Some improvements to IOS XRv OVA support

1.1.2 - 2014-09-24

Added

- Take advantage of QEMU 2.1 finally supporting the `streamOptimized` VMDK sub-format.
- Can now create new hardware items without an existing item of the same type (issue #4)

Changed

- Clearer documentation and logging messages (issue #8 and others)
- Now uses `versioneer` for automatic version numbering.

Fixed

- Fixed several Python 3 compatibility issues (issue #7 and others)

1.1.1 - 2014-08-19

Fixed

- Minor bug fixes to `cot deploy esxi`.

1.1.0 - 2014-07-29

Added

- `cot deploy esxi` subcommand by Kevin Keim (@kakeim), which uses `ovftool` to deploy an OVA to an ESXi vCenter server.

Changed

- Removed dependencies on `md5` / `md5sum` / `shasum` / `sha1sum` in favor of Python's `hashlib` module.
- Nicer formatting of `cot info` output

Fixed

- Miscellaneous fixes and code cleanup.

1.0.0 - 2014-06-27

Initial public release.

Contributing to COT

Please do contribute! We only have a few simple requirements for diffs and pull requests.

- *Follow coding guidelines*
- *Add automated unit tests*
- *Update documentation*
- *Add yourself as a contributor*
- *Open a pull request*

Follow coding guidelines

Logging level usage

ERROR Something is wrong (such as a violation of the OVF specification) but COT was able to attempt to recover. If recovery is not possible, you should raise an `Error` of appropriate type instead of logging an **ERROR** message.

WARNING Something unexpected or risky happened that the user needs a heads-up about. This includes cases where the software had to make an uncertain choice on its own due to lack of information from the user.

INFO Important status updates about normal operation of the software. As this is the lowest logging level enabled by default, you should keep the logs generated at this level relatively brief but meaningful.

VERBOSE Detailed information of interest to an advanced or inquisitive user.

DEBUG Highly detailed information only useful to a developer familiar with the code.

Coding style

We try to keep COT's code base compliant with Python coding standards including [PEP 8](#) and [PEP 257](#). We use the `flake8` and `Pylint` tools and their extension packages to verify this as part of our test automation. To run coding style analysis independently of the other test automation, you can run `tox -e flake8,pylint`, or you can install these tools and run them directly:

```
cot/$ sudo pip install --upgrade flake8
cot/$ sudo pip install --upgrade pydocstyle
cot/$ sudo pip install --upgrade flake8-docstrings
```

```
cot/$ sudo pip install --upgrade pep8-naming
cot/$ sudo pip install --upgrade mccabe
cot/$ flake8
./COT/ovf/item.py:229:1: C901 'OVFItem.value_replace_wildcards' is too complex (11)
./COT/ovf/item.py:603:1: C901 'OVFItem.generate_items' is too complex (11)
./COT/ovf/ovf.py:461:1: C901 'OVF.validate_hardware' is too complex (14)
```

```
cot/$ sudo pip install --upgrade pylint
cot/$ pylint COT
***** Module COT.ovf.item
E:331,24: Instance of 'list' has no 'split' member (no-member)
R:334,16: Redefinition of value type from list to tuple (redefined-variable-type)
R:603, 4: Too many branches (13/12) (too-many-branches)
***** Module COT.ovf.ovf
C: 1, 0: Too many lines in module (2646/2600) (too-many-lines)
R:177, 0: Too many public methods (76/74) (too-many-public-methods)
```

Fix any errors and warnings these tools report, and run again until no errors are reported.

Add automated unit tests

Whether adding new functionality or fixing a bug, **please** add appropriate unit test case(s) under `COT/tests/`, `COT/helpers/tests/`, or `COT/ovf/tests` (as appropriate) to cover your changes. Your changes **must** pass all existing and new automated test cases before your code will be accepted.

You can run the COT automated tests under a single Python version by running `python ./setup.py test`.

For full testing under all supported versions as well as verifying code coverage for your tests, you should install `tox` (pip install tox) and `coverage` (pip install coverage) then run `tox` from the COT directory:

```
cot/$ tox
...
py26 runtests: commands[0] | coverage run --append setup.py test --quiet
...
py27 runtests: commands[0] | coverage run --append setup.py test --quiet
...
py33 runtests: commands[0] | coverage run --append setup.py test --quiet
...
py34 runtests: commands[0] | coverage run --append setup.py test --quiet
...
py35 runtests: commands[0] | coverage run --append setup.py test --quiet
...
py36 runtests: commands[0] | coverage run --append setup.py test --quiet
...
pypy runtests: commands[0] | coverage run --append setup.py test --quiet
...
flake8 runtests: commands[0] | flake8
...
pylint runtests: commands[0] | pylint COT
...
docs runtests: commands[0] | sphinx-build -W -b html -d ...
...
stats runtests: commands[0] | coverage combine
stats runtests: commands[1] | coverage report -i
Name                               Stmt% Miss Branch BrPart  Cover
-----
COT/__init__.py                     5      0      0      0  100%
```

COT/add_disk.py	168	3	66	3	97%
COT/add_file.py	45	0	12	0	100%
COT/cli.py	254	15	95	9	93%
COT/data_validation.py	124	2	44	1	98%
COT/deploy.py	154	6	62	6	94%
COT/deploy_esxi.py	196	0	68	1	99%
COT/disks/__init__.py	23	0	10	0	100%
COT/disks/disk.py	56	1	20	1	97%
...					
COT/vm_description.py	166	4	4	0	98%
COT/vm_factory.py	26	0	4	0	100%
COT/xml_file.py	121	3	54	1	98%

TOTAL	5122	114	1908	105	97%

```

stats runtests: commands[2] | coverage html -i
_____ summary _____
  setup: commands succeeded
  py26: commands succeeded
  py27: commands succeeded
  py33: commands succeeded
  py34: commands succeeded
  py35: commands succeeded
  py36: commands succeeded
  pypy: commands succeeded
  flake8: commands succeeded
  pylint: commands succeeded
  docs: commands succeeded
  stats: commands succeeded
  congratulations :)

```

After running `tox` you can check the code coverage details by opening `htmlcov/index.html` in a web browser.

Update documentation

If you add or change any COT CLI or APIs, or add or remove any external dependencies, please update the relevant documentation.

Add yourself as a contributor

If you haven't contributed to COT previously, be sure to add yourself as a contributor in the `COPYRIGHT.txt` file.

Open a pull request

COT follows Vincent Driessen's [A successful Git branching model](#). As such, please submit feature enhancement and non-critical bugfix requests to merge into the `develop` branch rather than `master`.

Credits

We would like to thank:

- For evangelization, user feedback and bug reports:

- Sean Adams
- Arun Arunkumar
- Mark Coverdill
- Myles Dear
- Chandu Gutti
- Jeff Haag
- Jeff Loughridge
- Jonathan Muslow
- Scott O'Donnell
- Rick Ogg
- Anantha Padmanabha
- Keerthi Rawat
- David Rosenfeld
- Rafal Skorka
- Perumal Venkatesh
- John Withington

- For initial design review and comments:

- Andy Dalton
- Jusheng Feng
- Doug Gordon
- Lina Long
- Neil McGill
- Vinod Pandarinathan
- Rich Wellum

- For providing managerial support for the development and release of COT as open source:
 - Ray Romney
 - Sanjeev Tondale
 - Taskin Ucpinar
- [Rich Wellum](#), for creating “Build, Deploy, Execute OVA” (`bdeo.sh`), the precursor to COT.
- [Neil McGill](#), for creating and maintaining `fatdisk`
- Brian Somers, for creating and maintaining `vmdktool`

COT package reference

The below documents describe in depth the code structure and APIs of COT. These are not generally of interest to the end users of the COT script, but are provided for reference of developers wishing to integrate the COT package directly into their code. Package implementing the Common OVF Tool.

Virtual machine definition modules

<i>COT.vm_description</i>	Abstract superclass for reading, editing, and writing VMs.
<i>COT.vm_factory</i>	Factory for virtual machine objects.
<i>COT.vm_context_manager</i>	Context manager for virtual machine definitions.
<i>COT.xml_file</i>	Reading, editing, and writing XML files.

COT.vm_description module

Abstract superclass for reading, editing, and writing VMs.

<i>VMInitError</i>	Class representing errors encountered when trying to init/load a VM.
<i>VMDescription</i>	Abstract class for reading, editing, and writing VM definitions.

exception **VMInitError**

Bases: `exceptions.EnvironmentError`

Class representing errors encountered when trying to init/load a VM.

class **VMDescription** (*input_file*, *output_file=None*)

Bases: `object`

Abstract class for reading, editing, and writing VM definitions.

Properties

<i>input_file</i>	Data file to read in.
<i>output_file</i>	Filename that <i>write()</i> will output to.
<i>platform</i>	The Platform class object associated with this VM.
<i>config_profiles</i>	The list of supported configuration profiles.
<i>default_config_profile</i>	The name of the default configuration profile.
<i>environment_properties</i>	The array of environment properties.

Continued on next page

Table 8.3 – continued from previous page

<i>environment_transports</i>	The list of environment transport methods.
<i>networks</i>	The list of network names currently defined in this VM.
<i>network_descriptions</i>	The list of network descriptions currently defined in this VM.
<i>system_types</i>	List of virtual system type(s) supported by this virtual machine.
<i>version_short</i>	A short string describing the product version.
<i>version_long</i>	A long string describing the product version.

__init__ (*input_file*, *output_file*=None)

Read the given VM description file into memory.

Also creates a temporary directory as a working directory.

Parameters

- **input_file** (*str*) – Data file to read in.
- **output_file** (*str*) – File name to write to.
 - If this VM is read-only, (there will never be an output file) this value should be None
 - If the output filename is not yet known, use "" and subsequently set output when it is determined.

add_controller_device (*device_type*, *subtype*, *address*, *ctrl_item*=None)

Create a new IDE or SCSI controller, or update existing one.

Parameters

- **device_type** (*str*) – 'ide' or 'scsi'
- **subtype** (*str*) – Subtype such as 'virtio' (optional)
- **address** (*int*) – Controller address such as 0 or 1 (optional)
- **ctrl_item** (*object*) – Existing controller device to update (optional)

Returns *object* – New or updated controller device object

add_disk (*disk_repr*, *file_id*, *drive_type*, *disk*=None)

Add a new disk object to the VM or overwrite the provided one.

Parameters

- **disk_repr** (*DiskRepresentation*) – Disk file representation
- **file_id** (*str*) – Identifier string for the file/disk mapping
- **drive_type** (*str*) – 'harddisk' or 'cdrom'
- **disk** (*object*) – Existing disk object to overwrite

Returns *object* – New or updated disk object

add_disk_device (*drive_type*, *address*, *name*, *description*, *disk*, *file_obj*, *ctrl_item*, *disk_item*=None)

Add a new disk device to the VM or update the provided one.

Parameters

- **drive_type** (*str*) – 'harddisk' or 'cdrom'
- **address** (*str*) – Address on controller, such as "1:0" (optional)
- **name** (*str*) – Device name string (optional)
- **description** (*str*) – Description string (optional)

- **disk** (*object*) – Disk object to map to this device
- **file_obj** (*object*) – File object to map to this device
- **ctrl_item** (*object*) – Controller object to serve as parent
- **disk_item** (*object*) – Existing disk device to update instead of making a new device.

Returns *object* – New or updated disk device object.

add_file (*file_path*, *file_id*, *file_obj=None*, *disk=None*)

Add a new file object to the VM or overwrite the provided one.

Parameters

- **file_path** (*str*) – Path to file to add
- **file_id** (*str*) – Identifier string for the file in the VM
- **file_obj** (*object*) – Existing file object to overwrite
- **disk** (*object*) – Existing disk object referencing file.

Returns *object* – New or updated file object

check_sanity_of_disk_device (*disk*, *file_obj*, *disk_item*, *ctrl_item*)

Check if the given disk is linked properly to the other objects.

Parameters

- **disk** (*object*) – Disk object to validate
- **file_obj** (*object*) – File object which this disk should be linked to (optional)
- **disk_item** (*object*) – Disk device object which should link to this disk (optional)
- **ctrl_item** (*object*) – Controller device object which should link to the *disk_item*

Raises `ValueMismatchError` – if the given items are not linked properly.

config_file_to_properties (*file_path*, *user_configurable=None*)

Import each line of a text file into a configuration property.

Parameters

- **file_path** (*str*) – File name to import.
- **user_configurable** (*bool*) – Should the properties be configurable at deployment time by the user?

convert_disk_if_needed (*disk_image*, *kind*)

Convert the disk to a more appropriate format if needed.

Parameters

- **disk_image** (`DiskRepresentation`) – Disk to inspect and possibly convert
- **kind** (*str*) – Image type (harddisk/cdrom).

Returns `DiskRepresentation` – *disk_image*, if no conversion was required, or a new `DiskRepresentation` instance representing a converted image that has been created in *output_dir*.

create_configuration_profile (*pid*, *label*, *description*)

Create/update a configuration profile with the given ID.

Parameters

- **pid** (*str*) – Profile identifier

- **label** (*str*) – Brief descriptive label for the profile
- **description** (*str*) – Verbose description of the profile

create_network (*label*, *description*)

Define a new network with the given label and description.

Also serves to update the description of an existing network label.

Parameters

- **label** (*str*) – Brief label for the network
- **description** (*str*) – Verbose description of the network

delete_configuration_profile (*profile*)

Delete the configuration profile with the given ID.

Parameters **profile** (*str*) – Profile identifier

destroy ()

Clean up after ourselves.

Deletes `self.working_dir` and its contents.

classmethod detect_type_from_name (*filename*)

Check the given filename to see if it looks like a type we support.

Does not check file contents, as the given filename may not yet exist.

Parameters **filename** (*str*) – File name or path

Returns *str* – A string representing a recognized and supported type of file

Raises `ValueUnsupportedError` – if COT can't recognize the file type or doesn't know how to handle this file type.

find_device_location (*device*)

Find the controller type and address of a given device object.

Parameters **device** (*object*) – Hardware device object.

Returns *tuple* – (*type*, *address*), such as ("ide", "1:0").

find_empty_drive (*drive_type*)

Find a disk device that exists but contains no data.

Parameters **drive_type** (*str*) – Disk drive type, such as 'cdrom' or 'harddisk'

Returns *object* – Hardware device object, or None.

find_open_controller (*controller_type*)

Find the first open slot on a controller of the given type.

Parameters **controller_type** (*str*) – 'ide' or 'scsi'

Returns *tuple* – (*controller_device*, *address_string*) or (None, None)

get_common_subtype (*device_type*)

Get the sub-type common to all devices of the given type.

Parameters **device_type** (*str*) – Device type such as 'ide' or 'memory'.

Returns *str* – Subtype string common to all devices of this type, or None, if multiple such devices exist and they do not all have the same sub-type.

get_file_ref_from_disk (*disk*)

Get the file reference from the given opaque disk object.

Parameters `disk` (*object*) – Disk object to query

Returns *str* – String that can be used to identify the file associated with this disk

get_id_from_disk (*disk*)

Get the identifier string associated with the given Disk object.

Parameters `disk` (*object*) – Disk object

Returns *str* – Identifier string associated with this object

get_id_from_file (*file_obj*)

Get the file ID from the given opaque file object.

Parameters `file_obj` (*object*) – File object to query

Returns *str* – Identifier string associated with this object

get_nic_count (*profile_list*)

Get the number of NICs under the given profile(s).

Parameters `profile_list` (*list*) – Profile(s) of interest.

Returns *dict* – { `profile_name` : `nic_count` }

get_path_from_file (*file_obj*)

Get the file path from the given opaque file object.

Parameters `file_obj` (*object*) – File object to query

Returns *str* – Relative path to the file associated with this object

get_property_value (*key*)

Get the value of the given property.

Parameters `key` (*str*) – Property identifier

Returns *str* – Value of this property, or None

get_serial_connectivity (*profile*)

Get the serial port connectivity strings under the given profile.

Parameters `profile` (*str*) – Profile of interest.

Returns *list* – List of connectivity strings

get_serial_count (*profile_list*)

Get the number of serial ports under the given profile(s).

Parameters `profile_list` (*list*) – Change only the given profiles

Returns *dict* – { `profile_name` : `serial_count` }

info_string (*width=79, verbosity_option=None*)

Get a descriptive string summarizing the contents of this VM.

Parameters

- **width** (*int*) – Line length to wrap to where possible.
- **verbosity_option** (*str*) – ‘brief’, None (default), or ‘verbose’

Returns *str* – Wrapped, appropriately verbose string.

profile_info_string (*width=79, verbosity_option=None*)

Get a string summarizing available configuration profiles.

Parameters

- **width** (*int*) – Line length to wrap to if possible
- **verbosity_option** (*str*) – ‘brief’, None (default), or ‘verbose’

Returns *str* – Appropriately formatted and verbose string.

remove_file (*file_obj*, *disk=None*, *disk_drive=None*)

Remove the given file object from the VM.

Parameters

- **file_obj** (*object*) – File object to remove
- **disk** (*object*) – Disk object referencing file
- **disk_drive** (*object*) – Disk drive mapping file to a device

search_from_controller (*controller*, *address*)

From the controller type and device address, look for existing disk.

Parameters

- **controller** (*str*) – ‘ide’ or ‘scsi’
- **address** (*str*) – Device address such as ‘1:0’

Returns *tuple* – (file, disk, controller_device, disk_device), opaque objects of which any or all may be None

search_from_file_id (*file_id*)

From the given file ID, try to find any existing objects.

Parameters **file_id** (*str*) – File ID to search from

Returns *tuple* – (file, disk, controller_device, disk_device), opaque objects of which any or all may be None

search_from_filename (*filename*)

From the given filename, try to find any existing objects.

Parameters **filename** (*str*) – Filename to search from

Returns *tuple* – (file, disk, controller_device, disk_device), opaque objects of which any or all may be None

set_cpu_count (*cpus*, *profile_list*)

Set the number of CPUs.

Parameters

- **cpus** (*int*) – Number of CPUs
- **profile_list** (*list*) – Change only the given profiles

set_ide_subtype (*subtype*, *profile_list*)

Set the device subtype for the IDE controller(s).

Deprecated since version 1.5: Use `set_ide_subtypes()` instead.

Parameters

- **subtype** (*str*) – IDE subtype string
- **profile_list** (*list*) – Change only the given profiles

set_ide_subtypes (*type_list*, *profile_list*)

Set the device subtype list for the IDE controller(s).

Parameters

- **type_list** (*list*) – IDE subtype string list
- **profile_list** (*list*) – Change only the given profiles

set_memory (*megabytes, profile_list*)

Set the amount of RAM, in megabytes.

Parameters

- **megabytes** (*int*) – Memory value, in megabytes
- **profile_list** (*list*) – Change only the given profiles

set_nic_count (*count, profile_list*)

Set the given profile(s) to have the given number of NICs.

Parameters

- **count** (*int*) – number of NICs
- **profile_list** (*list*) – Change only the given profiles

set_nic_mac_addresses (*mac_list, profile_list*)

Set the MAC addresses for NICs under the given profile(s).

Note: If the length of `mac_list` is less than the number of NICs, will use the last entry in the list for all remaining NICs.

Parameters

- **mac_list** (*list*) – List of MAC addresses to assign to NICs
- **profile_list** (*list*) – Change only the given profiles

set_nic_names (*name_list, profile_list*)

Set the device names for NICs under the given profile(s).

Parameters

- **name_list** (*list*) – List of names to assign.
- **profile_list** (*list*) – Change only the given profiles

set_nic_networks (*network_list, profile_list*)

Set the NIC to network mapping for NICs under the given profile(s).

Note: If the length of `network_list` is less than the number of NICs, will use the last entry in the list for all remaining NICs.

Parameters

- **network_list** (*list*) – List of networks to map NICs to
- **profile_list** (*list*) – Change only the given profiles

set_nic_type (*nic_type, profile_list*)

Set the hardware type for NICs.

Deprecated since version 1.5: Use `set_nic_types()` instead.

Parameters

- **nic_type** (*str*) – NIC hardware type
- **profile_list** (*list*) – Change only the given profiles.

set_nic_types (*type_list*, *profile_list*)

Set the hardware type(s) for NICs.

Parameters

- **type_list** (*list*) – NIC hardware type(s)
- **profile_list** (*list*) – Change only the given profiles.

set_property_value (*key*, *value*, *user_configurable=None*, *property_type=None*, *label=None*, *description=None*)

Set the value of the given property (converting value if needed).

Parameters

- **key** (*str*) – Property identifier
- **value** (*object*) – Value to set for this property
- **user_configurable** (*bool*) – Should this property be configurable at deployment time by the user?
- **property_type** (*str*) – Value type - 'string' or 'boolean'
- **label** (*str*) – Brief explanatory label for this property
- **description** (*str*) – Detailed description of this property

Returns *str* – the (converted) value that was set.

set_scsi_subtype (*subtype*, *profile_list*)

Set the device subtype for the SCSI controller(s).

Deprecated since version 1.5: Use `set_scsi_subtypes()` instead.

Parameters

- **subtype** (*str*) – SCSI subtype string
- **profile_list** (*list*) – Change only the given profiles

set_scsi_subtypes (*type_list*, *profile_list*)

Set the device subtype list for the SCSI controller(s).

Parameters

- **type_list** (*list*) – SCSI subtype string list
- **profile_list** (*list*) – Change only the given profiles

set_serial_connectivity (*conn_list*, *profile_list*)

Set the serial port connectivity under the given profile(s).

Parameters

- **conn_list** (*list*) – List of connectivity strings
- **profile_list** (*list*) – Change only the given profiles

set_serial_count (*count*, *profile_list*)

Set the given profile(s) to have the given number of NICs.

Parameters

- **count** (*int*) – Number of serial ports
- **profile_list** (*list*) – Change only the given profiles

validate_hardware()

Check sanity of hardware properties for this VM/product/platform.

Returns *bool* – True if hardware is sane, False if not.

write()

Write the VM description to *output_file*, if any.

config_profiles

The list of supported configuration profiles.

If there are no profiles defined, returns an empty list. If there is a default profile, it will be first in the list.

default_config_profile

The name of the default configuration profile.

Returns *str* – Profile name or None if none are defined.

environment_properties

The array of environment properties.

Returns *list* – Array of dicts (one per property) with the keys "key", "value", "qualifiers", "type", "label", and "description".

environment_transports

The list of environment transport methods.

input_file

Data file to read in.

network_descriptions

The list of network descriptions currently defined in this VM.

networks

The list of network names currently defined in this VM.

output_file

Filename that *write()* will output to.

platform

The Platform class object associated with this VM.

GenericPlatform or a more specific subclass if recognized as such.

product_class

The product class identifier, such as com.cisco.csr1000v.

system_types

List of virtual system type(s) supported by this virtual machine.

verbosity_options = {'verbose': 2, 'brief': 0, None: 1}

version_long

A long string describing the product version.

version_short

A short string describing the product version.

COT.vm_factory module

Factory for virtual machine objects.

class VMFactory

Bases: `object`

Creates a `VMDescription` instance from a specified input file.

classmethod create (*input_file*, *output_file*)

Create an appropriate `VMDescription` subclass instance from a file.

Parameters

- **input_file** (*str*) – File to read VM description from
- **output_file** (*str*) – File to write to when finished (optional)

Raises

- `VMInitError` – if no appropriate class is identified
- `VMInitError` – if the selected subclass raises a `ValueUnsupportedError` while loading the file.

Returns `VMDescription` – Created object

COT.vm_context_manager module

Context manager for virtual machine definitions.

class VMContextManager (*input_file*, *output_file=None*)

Bases: `object`

Context manager for virtual machine definitions.

When the context manager exits, unless an error occurred, the virtual machine's `write()` method is called. Regardless of whether an error occurred, the virtual machine's `destroy()` method is then called.

Use as follows:

```
with VMContextManager(input_file, output_file) as vm:
    vm.foo()
    vm.bar()
```

For the parameters, see `VMDescription`.

__init__ (*input_file*, *output_file=None*)

Create a VM instance.

For the parameters, see `VMDescription`.

COT.xml_file module

Reading, editing, and writing XML files.

class XML (*xml_file*)

Bases: `object`

Class capable of reading, editing, and writing XML files.

`__init__(xml_file)`

Read the given XML file and store it in memory.

The memory representation is available as properties `tree` and `root`.

Parameters `xml_file` (*str*) – File path to read.

Raises

- `xml.etree.ElementTree.ParseError` – if parsing fails under Python 2.7 or later
- `xml.parsers.expat.ExpatError` – if parsing fails under Python 2.6

classmethod `add_child` (*parent*, *new_child*, *ordering=None*, *known_namespaces=None*)

Add the given child element under the given parent element.

Parameters

- **parent** (`xml.etree.ElementTree.Element`) – Parent element
- **new_child** (`xml.etree.ElementTree.Element`) – Child element to attach
- **ordering** (*list*) – (Optional) List describing the expected ordering of child tags under the parent; if a new child element is created, its placement under the parent will respect this sequence.
- **known_namespaces** (*list*) – (Optional) List of well-understood XML namespaces. If a new child is created, and `ordering` is given, any tag (new or existing) that is encountered but not accounted for in `ordering` will result in COT logging a warning **if and only if** the unaccounted-for tag is in a known namespace.

classmethod `find_all_children` (*parent*, *tag*, *attrib=None*)

Find all matching child elements under the specified parent element.

Parameters

- **parent** (`xml.etree.ElementTree.Element`) – Parent element
- **tag** (*iterable*) – Child tag string (or list of tags) to match on
- **attrib** (*dict*) – Child attributes to match on

Returns *list* – (Possibly empty) list of matching child Elements

classmethod `find_child` (*parent*, *tag*, *attrib=None*, *required=False*)

Find the unique child element under the specified parent element.

Parameters

- **parent** (`xml.etree.ElementTree.Element`) – Parent element
- **tag** (*str*) – Child tag to match on
- **attrib** (*dict*) – Child attributes to match on
- **required** (*boolean*) – Whether to raise an error if no child exists

Raises

- `LookupError` – if more than one matching child is found
- `KeyError` – if no matching child is found and `required` is True

Returns `xml.etree.ElementTree.Element` – Child element found, or None

static `get_ns` (*text*)

Get the namespace prefix from an XML element or attribute name.

Parameters `text` (*str*) – Element name or attribute name, such as “{<http://schemas.dmtf.org/ovf/envelope/1>}Element”.

Returns *str* – “” if no prefix is present, or a namespace prefix, such as “<http://schemas.dmtf.org/ovf/envelope/1>”.

classmethod `set_or_make_child`(*parent*, *tag*, *text=None*, *attrib=None*, *ordering=None*, *known_namespaces=None*)

Update or create a child element under the specified parent element.

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Parent element
- **tag** (*str*) – Child element text tag to find or create
- **text** (*str*) – Value to set the child’s text attribute to
- **attrib** (*dict*) – Dict of child attributes to match on while searching and set in the final child element
- **ordering** (*list*) – See `add_child()`
- **known_namespaces** (*list*) – See `add_child()`

Returns *xml.etree.ElementTree.Element* – New or updated child Element.

static `strip_ns`(*text*)

Remove a namespace prefix from an XML element or attribute name.

Parameters `text` (*str*) – Element name or attribute name, such as “{<http://schemas.dmtf.org/ovf/envelope/1>}Element”.

Returns *str* – Bare name, such as “Element”.

write_xml (*xml_file*)

Write pretty XML out to the given file.

Parameters `xml_file` (*str*) – Filename to write to

xml_reindent (*parent*, *depth*)

Recursively add indentation to XML to make it look nice.

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Current parent element
- **depth** (*int*) – How far down the rabbit hole we have recursed. Increments by 2 for each successive level of nesting.

root = None

Root *xml.etree.ElementTree.Element* instance of the tree.

tree = None

xml.etree.ElementTree.ElementTree describing this file.

register_namespace (*prefix*, *uri*)

Record a particular mapping between a namespace prefix and URI.

Parameters

- **prefix** (*str*) – Namespace prefix such as “ovf”
- **uri** (*str*) – Namespace URI such as “<http://schemas.dmtf.org/ovf/envelope/1>”

Command modules

<i>COT.submodule</i>	Parent classes for implementing COT subcommands.
<i>COT.add_disk</i>	Module for adding disks to VMs.
<i>COT.add_file</i>	Module for adding files to VM definitions.
<i>COT.deploy</i>	Module for deploying VM descriptions to a hypervisor to instantiate VMs.
<i>COT.deploy_esxi</i>	Module for deploying VMs to ESXi, vCenter, and vSphere.
<i>COT.edit_hardware</i>	Module for editing hardware details of a VM.
<i>COT.edit_product</i>	Module for editing product information in a VM description.
<i>COT.edit_properties</i>	Module for managing VM environment configuration properties.
<i>COT.help</i>	Provide ‘help’ keyword for COT CLI.
<i>COT.info</i>	Implements “info” subcommand.
<i>COT.inject_config</i>	Implements “inject-config” command.
<i>COT.install_helpers</i>	Implements “install-helpers” command.
<i>COT.remove_file</i>	Module for removing files from VM definitions.

COT.submodule module

Parent classes for implementing COT subcommands.

Classes

<i>COTGenericSubmodule</i>	Abstract interface for COT command submodules.
<i>COTReadOnlySubmodule</i>	Class for submodules that do not modify the OVF, such as ‘deploy’.
<i>COTSubmodule</i>	Class for submodules that read and write the OVF.

class **COTGenericSubmodule** (*ui*)

Bases: `object`

Abstract interface for COT command submodules.

Attributes: *vm*, *ui*

Note: Generally a command should not inherit directly from this class, but should instead subclass *COTReadOnlySubmodule* or *COTSubmodule* as appropriate.

__init__ (*ui*)

Instantiate this submodule with the given UI.

Parameters *ui* (`UI`) – User interface instance.

create_subparser ()

Add subparser for the CLI of this submodule.

destroy ()

Destroy any VM associated with this submodule.

finished ()

Do any final actions before being destroyed.

This class does nothing; subclasses may choose to do things like write their VM state out to a file.

ready_to_run ()

Check whether the module is ready to *run* ().

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

Raises `InvalidInputError` – if `ready_to_run()` reports False

ui = None

User interface instance (UI or subclass).

vm = None

Virtual machine description (VMDescription).

class COTReadOnlySubmodule(ui)

Bases: `COT.submodule.COTGenericSubmodule`

Class for submodules that do not modify the OVF, such as ‘deploy’.

Inherited attributes: `vm, ui`

Attributes: `package`

__init__(ui)

Instantiate this submodule with the given UI.

Parameters **ui** (UI) – User interface instance.

ready_to_run()

Check whether the module is ready to run().

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

package

VM description file to read from.

Calls `COT.vm_factory.VMFactory.create()` to instantiate `self.vm` from the provided file.

Raises `InvalidInputError` – if the file does not exist.

class COTSubmodule(ui)

Bases: `COT.submodule.COTGenericSubmodule`

Class for submodules that read and write the OVF.

Inherited attributes: `vm, ui`

Attributes: `package, output`

__init__(ui)

Instantiate this submodule with the given UI.

Parameters **ui** (UI) – User interface instance.

finished()

Write the current VM state out to disk if requested.

ready_to_run()

Check whether the module is ready to `run()`.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

If `output` was not previously set, automatically sets it to the value of `PACKAGE`.

Raises `InvalidInputError` – if `ready_to_run()` reports False

output

Output file for this submodule.

If the specified file already exists, will prompt the user (*confirm_or_die()*) to confirm overwriting the existing file.

package

VM description file to read (and possibly write).

Calls *COT.vm_factory.VMFactory.create()* to instantiate *self.vm* from the provided file.

Raises *InvalidInputError* – if the file does not exist.

COT.add_disk module

Module for adding disks to VMs.

Functions

<i>add_disk_worker</i>	Worker function for actually adding the disk.
<i>confirm_elements</i>	Get user confirmation of any risky or unusual operations.
<i>guess_controller_type</i>	If a controller type wasn't specified, try to guess from context.
<i>guess_drive_type_from_extension</i>	Guess the disk type (harddisk/cdrom) from the disk file name.
<i>search_for_elements</i>	Search for a unique set of objects based on the given criteria.
<i>validate_elements</i>	Validate any existing file, disk, controller item, and disk item.
<i>validate_controller_address</i>	Check validity of the given address string for the given controller.

Classes

<i>COTAddDisk</i>	Add or replace a disk in a virtual machine.
-------------------	---

class COTAddDisk (ui)

Bases: *COT.submodule.COTSubmodule*

Add or replace a disk in a virtual machine.

Inherited attributes: *ui*, *package*, *output*

Attributes: *disk_image*, *drive_type*, *file_id*, *controller*, *subtype*, *address*, *diskname*, *description*

__init__ (ui)

Instantiate this submodule with the given UI.

Parameters *ui* (*UI*) – User interface instance.

create_subparser ()

Create 'add-disk' CLI subparser.

ready_to_run ()

Check whether the module is ready to *run()*.

Returns *tuple* – (True, *ready_message*) or (False, *reason_why_not*)

run ()

Do the actual work of this submodule.

Raises *InvalidInputError* – if *ready_to_run()* reports False

address

Disk device address on controller (1:0, etc.).

Raises `InvalidInputError` – see `validate_controller_address()`

controller

Disk controller type (ide, scsi).

Raises `InvalidInputError` – see `validate_controller_address()`

description = None

Description of the disk.

disk_image

Disk image file to add to the VM.

Raises `InvalidInputError` – if the file does not exist.

diskname = None

Name string for the disk.

drive_type = None

Disk drive type ('harddisk' or 'cdrom').

file_id = None

File identifier to map disk to file.

subtype = None

Controller subtype, such as "virtio".

add_disk_worker (*vm*, *ui*, *disk_image*, *drive_type=None*, *file_id=None*, *controller=None*, *subtype=None*,
address=None, *diskname=None*, *description=None*)

Worker function for actually adding the disk.

All parameters except *vm*, *ui*, and *disk_image* are optional and will be automatically determined by COT if unspecified.

Parameters

- **vm** (`VMDescription`) – The virtual machine being edited.
- **ui** (`UI`) – User interface in effect.
- **disk_image** (`DiskRepresentation`) – Disk image to add to the VM.
- **drive_type** (`str`) – Disk drive type: 'cdrom' or 'harddisk'. If not specified, will be derived automatically from the *disk_image* file name extension.
- **file_id** (`str`) – Identifier of the disk file in the VM. If not specified, the VM will automatically derive an appropriate value.
- **controller** (`str`) – Disk controller type: 'ide' or 'scsi'. If not specified, will be derived from the *type* and the *platform* of the given *vm*.
- **subtype** (`str`) – Controller subtype ('virtio', 'lsilogic', etc.)
- **address** (`str`) – Disk device address on its controller (such as '1:0'). If this matches an existing disk device, that device will be overwritten. If not specified, the first available address not already occupied by an existing device will be selected.
- **diskname** (`str`) – Name for disk device
- **description** (`str`) – Description of disk device

confirm_elements (*vm, ui, file_obj, disk_image, disk_obj, disk_item, drive_type, controller, ctrl_item, subtype*)

Get user confirmation of any risky or unusual operations.

Parameters

- **vm** (*VMDescription*) – Virtual machine object
- **ui** (*UI*) – User interface object
- **file_obj** (*object*) – Known file object
- **disk_image** (*str*) – Filename or path for disk file
- **disk_obj** (*object*) – Known disk object
- **disk_item** (*object*) – Known disk device object
- **drive_type** (*str*) – “harddisk” or “cdrom”
- **controller** (*str*) – Controller type (“ide” or “scsi”)
- **ctrl_item** (*object*) – Known controller device object
- **subtype** (*str*) – Controller subtype (such as “virtio”)

guess_controller_type (*platform, ctrl_item, drive_type*)

If a controller type wasn’t specified, try to guess from context.

Parameters

- **platform** (*GenericPlatform*) – Platform class to guess controller for
- **ctrl_item** (*object*) – Any known controller object, or None
- **drive_type** (*str*) – “cdrom” or “harddisk”

Returns *str* – ‘ide’ or ‘scsi’

Raises `ValueUnsupportedError` – if `ctrl_item` is not None but is also not an IDE or SCSI controller device.

Examples

```
>>> from COT.platforms import GenericPlatform
>>> guess_controller_type(GenericPlatform, None, 'harddisk')
'ide'
```

guess_drive_type_from_extension (*disk_file_name*)

Guess the disk type (harddisk/cdrom) from the disk file name.

Parameters **disk_file_name** (*str*) – File name or file path.

Returns *str* – “cdrom” or “harddisk”

Raises `InvalidInputError` – if the disk type cannot be guessed.

Examples

```
>>> guess_drive_type_from_extension('/foo/bar.vmdk')
'harddisk'
>>> guess_drive_type_from_extension('baz.vmdk.iso')
'cdrom'
```

```
>>> try:
...     guess_drive_type_from_extension('/etc/os-release')
...     raise AssertionError("no exception raised")
... except InvalidInputError as e:
...     print(e)
Unable to guess disk drive type for file '/etc/os-release' from its extension ''.
Known extensions are ['.img', '.iso', '.qcow2', '.raw', '.vmdk']
Please specify '--type harddisk' or '--type cdrom'.
```

search_for_elements (*vm, disk_file, file_id, controller, address*)

Search for a unique set of objects based on the given criteria.

A disk is defined by up to four different sections in the OVF:

- File (references the actual disk image file)
- Disk (references the File, only used for HD not CD-ROM)
- Item (defines the SCSI/IDE controller)
- Item (defines the disk drive, links to controller and File or Disk)

For each of these four sections, we need to know whether to add a new one or overwrite an existing one. Depending on the user arguments, we can do this by as many as three different approaches:

- 1.Check whether the DISK_IMAGE file name matches an existing File in the OVF (and from there, find the associated Disk and Items)
- 2.Check whether the file-id matches an existing File and/or Disk in the OVF (and from there, find the associated Items)
- 3.Check whether controller type and/or device address match existing Items in the OVF (and from there, find the associated Disk and/or File)

Where it gets extra fun is if the user has specified more than one of the above arguments - in which case we need to make sure that all relevant approaches agree on what sections we’re talking about...

Parameters

- **vm** (*VMDescription*) – Virtual machine object
- **disk_file** (*str*) – Disk file name or path
- **file_id** (*str*) – File identifier
- **controller** (*str*) – controller type, “ide” or “scsi”
- **address** (*str*) – device address, such as “1:0”

Raises *ValueMismatchError* – if the criteria select a non-unique set.

Returns *tuple* – (file_object, disk_object, controller_item, disk_item)

validate_controller_address (*controller, address*)

Check validity of the given address string for the given controller.

Helper method for the controller/address setters.

Parameters

- **controller** (*str*) – ‘ide’ or ‘scsi’
- **address** (*str*) – A string like ‘0:0’ or ‘2:10’

Raises *InvalidInputError* – if the address/controller combo is invalid.

Examples

```
>>> validate_controller_address("ide", "0:0")
>>> try:
...     validate_controller_address("ide", "1:3")
... except InvalidInputError as e:
...     print(e)
IDE disk address must be between 0:0 and 1:1
>>> validate_controller_address("scsi", "1:3")
>>> try:
...     validate_controller_address("scsi", "4:0")
... except InvalidInputError as e:
...     print(e)
SCSI disk address must be between 0:0 and 3:15
```

validate_elements (*vm, file_obj, disk_obj, disk_item, ctrl_item, file_id, ctrl_type*)

Validate any existing file, disk, controller item, and disk item.

Raises `ValueMismatchError` – if the search criteria select a non-unique set.

Parameters

- **vm** (*VMDescription*) – Virtual machine object
- **file_obj** (*object*) – Known file object
- **disk_obj** (*object*) – Known disk object
- **disk_item** (*object*) – Known disk device object
- **ctrl_item** (*object*) – Known controller device object
- **file_id** (*str*) – File identifier string
- **ctrl_type** (*str*) – Controller type (“ide” or “scsi”)

COT.add_file module

Module for adding files to VM definitions.

`COTAddFile(ui)` Add a file (such as a README) to the package.

class `COTAddFile` (*ui*)

Bases: `COT.submodule.COTSubmodule`

Add a file (such as a README) to the package.

Inherited attributes: `ui, package, output`

Attributes: `file, file_id`

__init__ (*ui*)

Instantiate this submodule with the given UI.

Parameters **ui** (*UI*) – User interface instance.

create_subparser ()

Create ‘add-file’ CLI subparser.

ready_to_run ()

Check whether the module is ready to `run()`.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

Raises `InvalidInputError` – if `ready_to_run()` reports False

file

File to be added to the package.

Raises `InvalidInputError` – if the file does not exist.

file_id = None

File identifier string.

COT.deploy module

Module for deploying VM descriptions to a hypervisor to instantiate VMs.

Classes

<code>COTDeploy</code>	Semi-abstract class for submodules used to deploy a VM to a hypervisor.
<code>SerialConnection</code>	Generic class defining a serial port connection.

class COTDeploy(ui)

Bases: `COT.submodule.COTReadOnlySubmodule`

Semi-abstract class for submodules used to deploy a VM to a hypervisor.

Provides some baseline parameters and input validation that are expected to be common across all concrete subclasses.

Inherited attributes: `ui`, `package`,

Attributes: `generic_parser`, `parser`, `subparsers`, `hypervisor`, `configuration`, `username`, `password`, `power_on`, `vm_name`, `network_map`

__init__(ui)

Instantiate this submodule with the given UI.

Parameters `ui (UI)` – User interface instance.

create_subparser()

Create ‘deploy’ CLI subparser if it doesn’t already exist.

Note: Unlike most submodules, this one has subparsers of its own - ‘cot deploy PACKAGE <hypervisor>’ so subclasses of this module should call `super().create_subparser()` (to create the main ‘deploy’ subparser if it doesn’t already exist) then call `self.ui.add_parser(..., parent=self.subparsers, ...)` to add their own sub-subparser.

ready_to_run()

Check whether the module is ready to `run()`.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

configuration

VM configuration profile to use for deployment.

Raises `InvalidInputError` – if not a profile defined in the VM.

generic_parser = None

Generic parser object providing args that most subclasses will use.

Subclasses can call `self.subparsers.add_parser(parents=[self.generic_parser])` to automatically inherit this set of args

hypervisor

Hypervisor to deploy to.

Raises `InvalidInputError` – if not a recognized value.

network_map

Mapping of network names to networks.

password = None

Server login password.

power_on

Whether to automatically power on the VM after deployment.

serial_connection

Mapping of serial ports to various connection types.

subparsers = None

Subparser grouping for hypervisor-specific sub-subparsers.

Subclasses should generally have their `create_subparser()` implementations create their sub-subparsers with `parent=subparsers`.

username = None

Server login username.

vm_name = None

Name of the created virtual machine

class SerialConnection (*kind, value, options*)

Bases: `object`

Generic class defining a serial port connection.

__init__ (*kind, value, options*)

Construct a SerialConnection object of the given kind and value.

Parameters

- **kind** (*str*) – Connection type string, possibly in need of munging.
- **value** (*str*) – Connection value such as `‘/dev/ttyS0’` or `‘1.1.1.1:80’`
- **options** (*dict*) – Input options dictionary.

classmethod from_cli_string (*cli_string*)

Parse a string `‘kind:value[,opts]’` to build a SerialConnection.

Based on the QEMU CLI for serial ports.

Parameters **cli_string** (*str*) – String of the form `‘kind:value[,opts]’`

Returns *SerialConnection* – Created instance or None.

Raises `InvalidInputError` – if `cli_string` cannot be parsed

Examples

```
>>> str(SerialConnection.from_cli_string('/dev/ttyS0'))
'<SerialConnection kind: device value: /dev/ttyS0 options: {}>'
>>> str(SerialConnection.from_cli_string('tcp::22,server'))
"<SerialConnection kind: tcp value: :22 options: {'server': True}>"
>>> str(SerialConnection.from_cli_string('telnet://1.1.1.1:1111'))
'<SerialConnection kind: telnet value: 1.1.1.1:1111 options: {}>'
```

classmethod `validate_kind(kind)`

Validate the connection type string and munge it as needed.

Parameters `kind` (*str*) – Connection type string, possibly in need of munging.

Returns *str* – A valid type string

Raises `ValueUnsupportedError` – if kind is not recognized as valid

classmethod `validate_options(kind, value, options)`

Check that the given set of options are valid for this connection.

Parameters

- `kind` (*str*) – Validated ‘kind’ string.
- `value` (*str*) – Validated ‘value’ string. Currently unused.
- `options` (*dict*) – Input options dictionary.

Returns *dict* – Validated options

Raises `InvalidInputError` – if options are not valid.

classmethod `validate_value(kind, value)`

Check that the given value is valid for the given connection kind.

Parameters

- `kind` (*str*) – Connection type, valid per `validate_kind()`.
- `value` (*str*) – Connection value such as ‘/dev/ttyS0’ or ‘1.1.1.1:80’

Returns *str* – Munged value string.

Raises

- `InvalidInputError` – if value string is not recognized as valid
- `NotImplementedError` – if kind is not valid

kind = None

Connection type string

options = None

Dictionary of connection options.

value = None

Connection value such as ‘/dev/ttyS0’ or ‘1.1.1.1:80’

COT.deploy_esxi module

Module for deploying VMs to ESXi, vCenter, and vSphere.

Functions

`get_object_from_connection` Look up an object by name.

Classes

<code>COTDeployESXi</code>	Submodule for deploying VMs on ESXi and VMware vCenter/vSphere.
<code>SmarterConnection</code>	A smarter version of pyVmomi's SmartConnection context manager.
<code>PyVmomiVMReconfigSpec</code>	Context manager for reconfiguring an ESXi VM using PyVmomi.

class `COTDeployESXi(ui)`

Bases: `COT.deploy.COTDeploy`

Submodule for deploying VMs on ESXi and VMware vCenter/vSphere.

Inherited attributes: `ui`, `package`, `generic_parser`, `parser`, `subparsers`, `hypervisor`, `configuration`, `username`, `password`, `power_on`, `vm_name`, `network_map`, `serial_connection`

Attributes: `locator`, `datastore`, `ovftool_args`

`__init__(ui)`

Instantiate this submodule with the given UI.

Parameters `ui` (UI) – User interface instance.

`create_subparser()`

Add subparser for the CLI of this submodule.

This will create the shared parser, then create our own sub-subparser under subparsers.

`fixup_ovftool_args(ovftool_args, target)`

Make any needed modifications to the ovftool arguments.

Parameters

- **ovftool_args** (`list`) – Any existing ovftool arguments to begin with.
- **target** (`str`) – deployment target URI

Returns `list` – Updated ovftool arguments

`fixup_serial_ports()`

Use PyVmomi to create and configure serial ports for the new VM.

Raises `NotImplementedError` – If any `SerialConnection` in `serial_connection` has a `kind` other than 'tcp', 'telnet', or 'device'

`ready_to_run()`

Check whether the module is ready to `run()`.

Returns `tuple` – (True, ready_message) or (False, reason_why_not)

`run()`

Do the actual work of this submodule - deploying to ESXi.

Raises `InvalidInputError` – if `ready_to_run()` reports False

`datastore = None`

ESXi datastore to deploy to.

`host = None`

vSphere host to deploy to - set implicitly by self.locator.

locator

Target vSphere locator.

ovftool_args

List of CLI arguments to pass through to `ovftool`.

serial_connection

Mapping of serial ports to various connection types.

server = None

vCenter server or vSphere host - set implicitly by `self.locator`.

class `PyVmomiVMReconfigSpec` (*conn*, *vm_name*)

Bases: `object`

Context manager for reconfiguring an ESXi VM using PyVmomi.

__init__ (*conn*, *vm_name*)

Use the given name to look up a VM using the given connection.

Parameters

- **conn** (`SmarterConnection`) – Connection to ESXi.
- **vm_name** (*str*) – Virtual machine name.

class `SmarterConnection` (*ui*, *host*, *user*, *pwd*, *port=443*)

Bases: `pyVim.connect.SmartConnection`

A smarter version of pyVmomi's `SmartConnection` context manager.

__init__ (*ui*, *host*, *user*, *pwd*, *port=443*)

Create a connection to the given server.

Parameters **ui** (`UI`) – User interface instance.

For the other parameters, see `pyVim.connect.SmartConnection`

static `unwrap_connection_error` (*outer_e*)

Extract inner attributes from a `ConnectionError`.

`ConnectionError` often wraps another exception with more context; this function dives inside the `ConnectionError` to find that context.

Parameters **outer_e** (`ConnectionError`) – `ConnectionError` to unwrap

Returns *tuple* – extracted (errno, inner_message)

get_object_from_connection (*conn*, *vimtype*, *name*)

Look up an object by name.

Parameters

- **conn** (`SmarterConnection`) – Connection to ESXi.
- **vimtype** (*object*) – currently only `vim.VirtualMachine`
- **name** (*str*) – Name of the object to look up.

Returns *object* – Located object

`COT.edit_hardware` module

Module for editing hardware details of a VM.

Functions

<code>expand_list_wildcard</code>	Expand a list containing a wildcard to the desired length.
<code>guess_list_wildcard</code>	Inverse of <code>expand_list_wildcard()</code> .

Classes

<code>COTEditHardware</code>	Edit hardware information (CPUs, RAM, NICs, etc.).
------------------------------	--

class `COTEditHardware(ui)`

Bases: `COT.submodule.COTSubmodule`

Edit hardware information (CPUs, RAM, NICs, etc.).

Inherited attributes: `ui`, `package`, `output`

Attributes: `profiles`, `delete_all_other_profiles`, `cpus`, `memory`, `nics`, `nic_types`, `mac_addresses_list`, `nic_networks`, `nic_names`, `network_descriptions`, `serial_ports`, `serial_connectivity`, `scsi_subtypes`, `ide_subtypes`, `virtual_system_type`

`__init__(ui)`

Instantiate this submodule with the given UI.

Parameters `ui` (UI) – User interface instance.

`create_subparser()`

Create ‘edit-hardware’ CLI subparser.

`ready_to_run()`

Check whether the module is ready to `run()`.

Returns `tuple` – (True, `ready_message`) or (False, `reason_why_not`)

`run()`

Do the actual work of this submodule.

Raises `InvalidInputError` – if `ready_to_run()` reports False

MEMORY_REGEX = ‘`^\\s*(\\d+)\\s*([mMgG])?i?[bB]?\\s*$`’

cpus

Number of CPUs to set.

delete_all_other_profiles = None

Delete all profiles other than those set in `profiles`.

ide_subtype

IDE controller subtype string to set.

Deprecated since version 1.5: Use `ide_subtypes` instead.

ide_subtypes

IDE controller subtype string(s) to set.

mac_addresses_list = None

List of MAC addresses to set.

memory

Amount of RAM (in megabytes) to set.

network_descriptions = None

List of network description strings.

Can use wildcards as described in `expand_list_wildcard()`.

nic_names = None

List of NIC name strings.

Can use wildcards as described in `expand_list_wildcard()`.

nic_networks = None

List of NIC-to-network mappings.

Can use wildcards as described in `expand_list_wildcard()`.

nic_type

NIC type string to set.

Deprecated since version 1.5: Use `nic_types` instead.

nic_types

List of NIC type strings to set.

nics

Number of NICs to set.

profiles = None

Configuration profile(s) to edit.

scsi_subtype

SCSI controller subtype string to set.

Deprecated since version 1.5: Use `scsi_subtypes` instead.

scsi_subtypes

SCSI controller subtype string(s) to set.

serial_connectivity = None

List of serial connection strings.

serial_ports

Serial port count to set.

virtual_system_type = None

Virtual system type

expand_list_wildcard (*name_list*, *length*, *quiet=False*)

Expand a list containing a wildcard to the desired length.

Parameters

- **name_list** (*list*) – List of names to assign, or None
- **length** (*list*) – Length to expand to
- **quiet** (*bool*) – Silence usual log messages generated by this function.

Returns *list* – Expanded list, or empty list if *name_list* is None or empty.

Since various items (NIC names, network names, etc.) are often named or numbered sequentially, we provide this API to allow the user to specify a wildcard value to permit automatically expanding a list of input strings to the desired length. The syntax for the wildcard option is { followed by a number (indicating the starting index for the name) followed by }.

Examples

```
>>> expand_list_wildcard(None, 3)
[]
>>> expand_list_wildcard(["eth{0}"], 3)
['eth0', 'eth1', 'eth2']
>>> expand_list_wildcard(["mgmt0", "eth{10}"], 4)
['mgmt0', 'eth10', 'eth11', 'eth12']
```

guess_list_wildcard(*known_values*)

Inverse of `expand_list_wildcard()`. Guess the wildcard for a list.

Parameters **known_values** (*list*) – Values to guess from

Returns *list* – Guessed wildcard list, or None if unable to guess

Examples

```
>>> guess_list_wildcard(['foo', 'bar', 'baz'])
>>> guess_list_wildcard(['foo1', 'foo2', 'foo3'])
['foo{1}']
>>> guess_list_wildcard(['foo', 'bar', 'baz3', 'baz4', 'baz5'])
['foo', 'bar', 'baz{3}']
>>> guess_list_wildcard(['Eth0/1', 'Eth0/2', 'Eth0/3'])
['Eth0/{1}']
>>> guess_list_wildcard(['Eth0/0', 'Eth1/0', 'Eth2/0'])
['Eth{0}/0']
>>> guess_list_wildcard(['fake1', 'fake2', 'real4', 'real5'])
['fake1', 'fake2', 'real{4}']
```

COT.edit_product module

Module for editing product information in a VM description.

Classes

<code>COTEditProduct</code>	Edit product, vendor, and version information strings.
-----------------------------	--

class COTEditProduct (*ui*)

Bases: `COT.submodule.COTSubmodule`

Edit product, vendor, and version information strings.

Inherited attributes: `ui`, `package`, `output`

Attributes: `product_class` `product` `vendor` `version`, `full_version` `product_url` `vendor_url` `application_url`

__init__(*ui*)

Instantiate this submodule with the given UI.

Parameters **ui** (`UI`) – User interface instance.

create_subparser()

Create ‘edit-product’ CLI subparser.

ready_to_run()

Check whether the module is ready to *run()*.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

Raises *InvalidInputError* – if *ready_to_run()* reports False

application_url = None

Application URL string.

full_version = None

Long version string.

product = None

Product description string.

product_class = None

Product class identifier.

product_url = None

Product URL string.

vendor = None

Vendor string.

vendor_url = None

Vendor URL string.

version = None

Short version string.

COT.edit_properties module

Module for managing VM environment configuration properties.

Classes

<i>COTEditProperties</i>	Edit OVF environment XML properties.
--------------------------	--------------------------------------

class COTEditProperties(ui)

Bases: *COT.submodule.COTSubmodule*

Edit OVF environment XML properties.

Inherited attributes: ui, package, output

Attributes: *config_file*, *properties*, *transports*, *user_configurable*

__init__(ui)

Instantiate this submodule with the given UI.

Parameters *ui* (*UI*) – User interface instance.

create_subparser()

Create 'edit-properties' CLI subparser.

edit_properties_interactive()

Present an interactive UI for the user to edit properties.

ready_to_run()

Check whether the module is ready to `run()`.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

Raises `InvalidInputError` – if `ready_to_run()` reports False

config_file

Path to plaintext file to read configuration lines from.

Raises `InvalidInputError` – if the file does not exist.

descriptions = None

List of description strings to set for updated properties.

labels = None

List of label strings to set for the properties being updated.

properties

List of property (key, value, type) tuples to update.

Properties may also be set from strings (such as by CLI) with the syntax `<key>[=<value>][+<type>]`.

Examples

```
>>> from COT.ui_shared import UI
>>> i = COTEditProperties(UI())
>>> i.properties
[]
>>> i.properties = [
...     "no_value",
...     "key=value",
...     "string_type+string",
...     "full-type=yes+boolean",
... ]
>>> print("\n".join([str(p) for p in i.properties]))
('no_value', None, None)
('key', 'value', None)
('string_type', None, 'string')
('full-type', 'yes', 'boolean')
>>> i.properties = [
...     "ssh=autopubkey=ssh-rsa AA...q+t0...Tuw== root@MASTER",
...     "tricky+=foo",
...     "tricky_value=++foo==++",
...     "trickiest=bar+foo=hello+boolean",
... ]
>>> print("\n".join([str(p) for p in i.properties]))
('ssh', 'autopubkey=ssh-rsa AA...q+t0...Tuw== root@MASTER', None)
('tricky', '', 'foo')
('tricky_value', '++foo==++', None)
('trickiest', 'bar+foo=hello', 'boolean')
```

transports

Transport mechanism(s) for environment properties.

user_configurable = None

Value to set the user_configurable flag on properties we edit.

COT.help module

Provide ‘help’ keyword for COT CLI.

class COTHelp (ui)

Bases: *COT.submodule.COTGenericSubmodule*

Provide ‘help <subcommand>’ syntax.

Inherited attributes: *ui*

Attributes: *subcommand*

__init__ (ui)

Instantiate this submodule with the given UI.

Parameters *ui* (*UI*) – User interface instance.

create_subparser ()

Create ‘help’ CLI subparser.

run ()

Display the help menu for the specified subcommand.

subcommand

CLI subcommand to give help for.

If None, then help will be displayed for the COT global parser.

COT.info module

Implements “info” subcommand.

class COTInfo (ui)

Bases: *COT.submodule.COTGenericSubmodule*

Display VM information string.

Inherited attributes: *ui*

Attributes: *package_list, verbosity*

__init__ (ui)

Instantiate this submodule with the given UI.

Parameters *ui* (*UI*) – User interface instance.

create_subparser ()

Create ‘info’ CLI subparser.

ready_to_run ()

Check whether the module is ready to *run ()*.

Returns *tuple* – (True, ready_message) or (False, reason_why_not)

run ()

Do the actual work of this submodule.

Raises *InvalidInputError* – if *ready_to_run ()* reports False

package_list

List of VM definitions to get information for.

verbosity

Verbosity of information displayed.

COT.inject_config module

Implements “inject-config” command.

class COTInjectConfig(ui)

Bases: *COT.submodule.COTSubmodule*

Wrap configuration file(s) into a disk image embedded into the VM.

Inherited attributes: ui, package, output

Attributes: *config_file*, *secondary_config_file*, *extra_files*

__init__(ui)

Instantiate this submodule with the given UI.

Parameters **ui** (UI) – User interface instance.

create_subparser()

Create ‘inject-config’ CLI subparser.

ready_to_run()

Check whether the module is ready to *run()*.

Returns *tuple* – (True, *ready_message*) or (False, *reason_why_not*)

run()

Do the actual work of this submodule.

Raises

- *InvalidInputError* – if *ready_to_run()* reports False
- *ValueUnsupportedError* – if the BOOTSTRAP_DISK_TYPE of the associated VM’s *platform* is not ‘cdrom’ or ‘harddisk’
- *LookupError* – if unable to find a disk drive device to inject the configuration into.

config_file

Primary configuration file.

Raises

- *InvalidInputError* – if the file does not exist
- *InvalidInputError* – if the platform described by package doesn’t support configuration files.

extra_files

Additional files to be embedded as-is.

Raises *InvalidInputError* – if any file in the list does not exist

secondary_config_file

Secondary configuration file.

Raises

- *InvalidInputError* – if the file does not exist

- `InvalidInputError` – if the platform described by `package` doesn't support secondary configuration files.

COT.install_helpers module

Implements “install-helpers” command.

class `COTInstallHelpers` (*ui*)

Bases: `COT.submodule.COTGenericSubmodule`

Install all helper tools that COT requires.

Inherited attributes: `ui`, `package`, `output`

__init__ (*ui*)

Instantiate this submodule with the given UI.

Parameters `ui` (`UI`) – User interface instance.

create_subparser ()

Create ‘install-helpers’ CLI subparser.

install_helper (*helper*)

Install the given helper module.

Parameters `helper` (`Helper`) – Helper module to install.

Returns *tuple* – (result, message)

manpages_helper ()

Verify or install COT’s manual pages.

Returns *tuple* – (result, message)

run ()

Verify all helper tools and install any that are missing.

guess_manpath ()

Guess the directory path where man pages should be installed.

install_manpages (*man_dir*)

Install COT’s manual pages.

Parameters `man_dir` (*str*) – Base directory where manpages should be installed.

Returns *tuple* – (result, message)

verify_manpages (*man_dir*)

Verify installation of COT’s manual pages.

Parameters `man_dir` (*str*) – Base directory where manpages should be found.

Returns *tuple* – (result, message)

COT.remove_file module

Module for removing files from VM definitions.

<code>COTRemoveFile(ui)</code>	Remove a file (such as a README) from the package.
--------------------------------	--

class `COTRemoveFile` (*ui*)

Bases: *COT.submodule.COTSubmodule*

Remove a file (such as a README) from the package.

Inherited attributes:

ui, package, output

Attributes: *file_path*, *file_id*

__init__ (*ui*)

Instantiate this submodule with the given UI.

Parameters *ui* (*UI*) – User interface instance.

create_subparser ()

Create ‘remove-file’ CLI subparser.

ready_to_run ()

Check whether the module is ready to *run* ().

Returns *tuple* – (True, *ready_message*) or (False, *reason_why_not*)

run ()

Do the actual work of this submodule.

Raises *InvalidInputError* – if *ready_to_run* () reports False

file_id = None

File identifier to be removed from the package.

file_path = None

File name or path to be removed from the package.

Helper library modules

<i>COT.data_validation</i>	Various helpers for data sanity checks.
<i>COT.file_reference</i>	Wrapper classes to abstract away differences between file sources.
<i>COT.platforms</i>	Package for identifying guest platforms and handling platform differences.

COT.data_validation module

Various helpers for data sanity checks.

Exceptions

<i>InvalidInputError</i>	Miscellaneous error during validation of user input.
<i>ValueMismatchError</i>	Values which were expected to be equal turned out to be not equal.
<i>ValueUnsupportedError</i>	An unsupported value was provided.
<i>ValueTooLowError</i>	A numerical input was less than the lowest supported value.
<i>ValueTooHighError</i>	A numerical input was higher than the highest supported value.

Functions

<i>alphanumeric_split</i>	Split the key into a list of [text, int, text, int, ..., text].
<i>canonicalize_helper</i>	Try to find a mapping of input to output.
Continued on next page	

Table 8.19 – continued from previous page

<code>canonicalize_ide_subtype</code>	Try to convert the given IDE controller string to a canonical form.
<code>canonicalize_nic_subtype</code>	Try to convert the given NIC subtype string to a canonical form.
<code>canonicalize_scsi_subtype</code>	Try to convert the given SCSI controller string to a canonical form.
<code>check_for_conflict</code>	Make sure the list does not contain references to more than one object.
<code>device_address</code>	Parser helper function for device address arguments.
<code>file_checksum</code>	Get the checksum of the given file.
<code>mac_address</code>	Parser helper function for MAC address arguments.
<code>match_or_die</code>	Make sure “first” and “second” are equal or raise an error.
<code>natural_sort</code>	Sort the given list “naturally” rather than in ASCII order.
<code>no_whitespace</code>	Parser helper function for arguments not allowed to contain whitespace.
<code>non_negative_int</code>	Parser helper function for integer arguments that must be 0 or more.
<code>positive_int</code>	Parser helper function for integer arguments that must be 1 or more.
<code>to_string</code>	Get string representation of an object, special-case for XML Element.
<code>validate_int</code>	Parser helper function for validating integer arguments in a range.
<code>truth_value</code>	Parser helper function for truth values like ‘0’, ‘y’, or ‘false’.

Constants

<code>NIC_TYPES</code>	List of NIC type strings recognized as canonical.
------------------------	---

exception `InvalidInputError`

Bases: `exceptions.ValueError`

Miscellaneous error during validation of user input.

exception `ValueMismatchError`

Bases: `exceptions.ValueError`

Values which were expected to be equal turned out to be not equal.

exception `ValueTooHighError` (*value_type*, *actual_value*, *expected_value*)

Bases: `COT.data_validation.ValueUnsupportedError`

A numerical input was higher than the highest supported value.

Parameters

- **`value_type`** (*str*) – descriptive string
- **`actual_value`** (*int*) – invalid value that was provided
- **`expected_value`** (*int*) – maximum supported value

exception `ValueTooLowError` (*value_type*, *actual_value*, *expected_value*)

Bases: `COT.data_validation.ValueUnsupportedError`

A numerical input was less than the lowest supported value.

Parameters

- **`value_type`** (*str*) – descriptive string
- **`actual_value`** (*int*) – invalid value that was provided
- **`expected_value`** (*int*) – minimum supported value

exception `ValueUnsupportedError` (*value_type*, *actual_value*, *expected_value*)

Bases: `COT.data_validation.InvalidInputError`

An unsupported value was provided.

Parameters

- **value_type** (*str*) – descriptive string
- **actual_value** (*str*) – invalid value that was provided
- **expected_value** (*object*) – expected/valid value(s) (item or list)

__init__ (*value_type, actual_value, expected_value*)
Create an instance of this class.

alphanum_split (*key*)

Split the key into a list of [text, int, text, int, ..., text].

Parameters **key** (*str*) – String to split.

Returns *list* – List of tokens

Examples

```
>>> alphanum_split("hello1world27")
['hello', 1, 'world', 27, '']
>>> alphanum_split("1istheloneliestnumber")
['', 1, 'istheloneliestnumber']
```

canonicalize_helper (*label, user_input, mappings, re_flags=0*)

Try to find a mapping of input to output.

Parameters

- **label** (*str*) – Label to use in any error raised
- **user_input** (*str*) – User-provided string
- **mappings** (*list*) – List of (*expr*, *canonical*) pairs for mapping.
- **re_flags** (*int*) – `re.IGNORECASE`, etc. if desired

Returns *str* – The canonical string

Raises *ValueUnsupportedError* – If no *expr* in *mappings* matches the given *user_input*.

canonicalize_ide_subtype (*subtype*)

Try to convert the given IDE controller string to a canonical form.

Parameters **subtype** (*str*) – User-provided string

Returns

str – The canonical string, one of:

- `PIIX4`
- `virtio`

Raises *ValueUnsupportedError* – If the canonical string cannot be determined

Examples

```
>>> canonicalize_ide_subtype('VirtIO')
'virtio'
>>> canonicalize_ide_subtype('PIIX4')
'PIIX4'
>>> try:
...     canonicalize_ide_subtype('usb')
... except ValueError as e:
...     print(e)
Unsupported value 'usb' for IDE controller subtype...
```

canonicalize_nic_subtype (*subtype*)

Try to convert the given NIC subtype string to a canonical form.

Parameters *subtype* (*str*) – User-provided string

Returns *str* – The canonical string, one of *NIC_TYPES*

Raises *ValueUnsupportedError* – If the canonical string cannot be determined

Examples

```
>>> canonicalize_nic_subtype('e1000')
'E1000'
>>> canonicalize_nic_subtype('vmxnet 3')
'VMXNET3'
>>> try:
...     canonicalize_nic_subtype('foobar')
... except ValueError as e:
...     print(e)
Unsupported value 'foobar' for NIC subtype ...
```

See also:

`COT.platforms.GenericPlatform.validate_nic_type()`

canonicalize_scsi_subtype (*subtype*)

Try to convert the given SCSI controller string to a canonical form.

Parameters *subtype* (*str*) – User-provided string

Returns

str – The canonical string, one of:

- buslogic
- lsilogic
- lsilogicsas
- virtio
- VirtualSCSI

Raises *ValueUnsupportedError* – If the canonical string cannot be determined

Examples

```
>>> canonicalize_scsi_subtype('LSI Logic')
'lsilogic'
>>> canonicalize_scsi_subtype('VirtIO')
'virtio'
>>> try:
...     canonicalize_scsi_subtype('baz')
... except ValueError as e:
...     print(e)
Unsupported value 'baz' for SCSI controller subtype...
```

check_for_conflict (*label*, *li*)

Make sure the list does not contain references to more than one object.

Parameters

- **label** (*str*) – Descriptive label to be used if an error is raised
- **li** (*list*) – List of object references (which may include None)

Raises *ValueMismatchError* – if references differ

Returns *object* – the object or None

Examples

```
>>> check_for_conflict("example", ['foo', None, 'foo'])
'foo'
>>> try:
...     check_for_conflict("conflict", [None, 'foo', 'bar'])
... except ValueMismatchError as e:
...     print(e)
Found multiple candidates for the conflict:
foo
...and...
bar
Please correct or clarify your search parameters.
```

device_address (*string*)

Parser helper function for device address arguments.

Validate string is an appropriately formed device address such as '1:0'.

Parameters **string** (*str*) – String to validate

Raises *InvalidInputError* – if string is not a well-formatted device address

Returns *str* – Validated string (with leading/trailing whitespace stripped)

Examples

```
>>> device_address(" 1:0\n")
'1:0'
>>> try:
...     device_address("1:0:1")
... except InvalidInputError as e:
...     print(e)
'1:0:1' is not a valid device address
```

file_checksum (*path_or_obj*, *checksum_type*)

Get the checksum of the given file.

Parameters

- **path_or_obj** (*str*) – File path to checksum OR an opened file object
- **checksum_type** (*str*) – Supported values are ‘md5’ and ‘sha1’.

Returns *str* – Hexadecimal file checksum

mac_address (*string*)

Parser helper function for MAC address arguments.

Validate whether a string is a valid MAC address. Recognized formats are:

- XX:XX:XX:XX:XX:XX
- XX-XX-XX-XX-XX-XX
- XXXX.XXXX.XXXX

Parameters **string** (*str*) – String to validate

Raises *InvalidInputError* – if string is not a valid MAC address

Returns *str* – Validated string (with leading/trailing whitespace stripped)

match_or_die (*first_label*, *first*, *second_label*, *second*)

Make sure “first” and “second” are equal or raise an error.

Parameters

- **first_label** (*str*) – Descriptive label for *first*
- **first** (*object*) – First object to compare
- **second_label** (*str*) – Descriptive label for *second*
- **second** (*object*) – Second object to compare

Raises *ValueMismatchError* – if *first* != *second*

Examples

```
>>> try:
...     match_or_die("old", 1, "new", 2)
... except ValueError as e:
...     print(e)
old 1 does not match new 2
```

natural_sort (*l*)

Sort the given list “naturally” rather than in ASCII order.

E.g, “10” comes after “9” rather than between “1” and “2”.

See also http://nedbatchelder.com/blog/200712/human_sorting.html

Parameters **l** (*list*) – List to sort

Returns *list* – Sorted list

Examples

```
>>> natural_sort(["Eth3", "Eth1", "Eth10", "Eth2"])
['Eth1', 'Eth2', 'Eth3', 'Eth10']
>>> natural_sort(["3rd", "1st", "10th", "101st"])
['1st', '3rd', '10th', '101st']
```

no_whitespace (*string*)

Parser helper function for arguments not allowed to contain whitespace.

Parameters **string** (*str*) – String to validate

Raises *InvalidInputError* – if string contains internal whitespace

Returns *str* – Validated string (with leading/trailing whitespace stripped)

Examples

```
>>> no_whitespace("    hello    ")
'hello'
>>> try:
...     no_whitespace('hello world')
... except InvalidInputError as e:
...     print(e)
'hello world' contains invalid whitespace
```

non_negative_int (*string*)

Parser helper function for integer arguments that must be 0 or more.

Alias for *validate_int()* setting minimum to 0.

Parameters **string** (*str*) – String to validate.

Returns *int* – Validated integer value

Raises

- *ValueUnsupportedError* – if string can't be converted to int
- *ValueTooLowError* – if value is less than 0

Examples

```
>>> non_negative_int('0')
0
>>> non_negative_int('1000')
1000
>>> try:
...     non_negative_int('-1')
... except ValueTooLowError as e:
...     print(e)
Value '-1' for input is too low - must be at least 0
```

positive_int (*string*)

Parser helper function for integer arguments that must be 1 or more.

Alias for *validate_int()* setting minimum to 1.

Parameters **string** (*str*) – String to validate.

Returns *int* – Validated integer value

Raises

- *ValueUnsupportedError* – if string can't be converted to int
- *ValueTooLowError* – if value is less than 1

Examples

```
>>> positive_int('1')
1
>>> try:
...     positive_int('0')
... except ValueTooLowError as e:
...     print(e)
Value '0' for input is too low - must be at least 1
```

to_string (*obj*)

Get string representation of an object, special-case for XML Element.

Parameters *obj* (*object*) – Object to represent as a string.

Returns *str* – string representation

Examples

```
>>> to_string("Hello")
'Hello'
>>> to_string(27.5)
'27.5'
>>> e = ET.Element('hello', attrib={'key': 'value'})
>>> print(e)
<Element ...hello... at ...>
>>> print(to_string(e))
<hello key="value" />
```

truth_value (*value*)

Parser helper function for truth values like '0', 'y', or 'false'.

Makes use of `distutils.util.strtobool()`, but returns True/False rather than 1/0.

Parameters *value* (*str*) – String to parse/validate

Returns *bool* – True or False

Raises *ValueUnsupportedError* – if the value can't be parsed to a boolean.

Examples

```
>>> truth_value('y')
True
>>> truth_value('false')
False
>>> truth_value(True)
True
>>> try:
```

```
...     truth_value('foo')
... except ValueUnsupportedError as e:
...     print(e)
Unsupported value 'foo' for truth value - expected ['y', ...
```

validate_int (*string*, *minimum=None*, *maximum=None*, *label='input'*)

Parser helper function for validating integer arguments in a range.

Parameters

- **string** (*str*) – String to convert to an integer and validate
- **minimum** (*int*) – Minimum valid value (optional)
- **maximum** (*int*) – Maximum valid value (optional)
- **label** (*str*) – Label to include in any errors raised

Returns *int* – Validated integer value

Raises

- *ValueUnsupportedError* – if string can't be converted to int
- *ValueTooLowError* – if value is less than minimum
- *ValueTooHighError* – if value is more than maximum

Examples

```
>>> validate_int('1')
1
>>> try:
...     validate_int('foo', label='x')
... except ValueUnsupportedError as e:
...     print(e)
Unsupported value 'foo' for x - expected integer
>>> try:
...     validate_int('100', label='x', maximum=10)
... except ValueTooHighError as e:
...     print(e)
Value '100' for x is too high - must be at most 10
```

NIC_TYPES = ['E1000e', 'E1000', 'PCNet32', 'virtio', 'VMXNET3']

List of NIC type strings recognized as canonical.

COT.file_reference module

Wrapper classes to abstract away differences between file sources.

class FileInTAR (*tarfile_path*, *filename*)

Bases: *object*

Wrapper for a file inside a TAR archive or OVA.

__init__ (*tarfile_path*, *filename*)

Create a reference to a file contained in a TAR archive.

Parameters

- **tarfile_path** (*str*) – Path to TAR archive to read

- **filename** (*str*) – File name in the TAR archive.

Raises `IOError` – if `tarfile_path` doesn't reference a TAR file, or the TAR file does not contain `filename`.

add_to_archive (*tarf*)

Copy this file into the given tarfile object.

Parameters **tarf** (*tarfile.TarFile*) – Add this file to that archive.

close ()

Close the file object previously opened.

copy_to (*dest_dir*)

Extract this file to the given destination directory.

Parameters **dest_dir** (*str*) – Destination directory or filename.

open (*mode*)

Open the TAR and return a reference to the relevant file object.

Parameters **mode** (*str*) – Only 'r' and 'rb' modes are supported.

Returns *file* – File object

Raises `ValueError` – if mode is not valid.

exists

True if the file exists in the TAR archive, else False.

size

The size of this file in bytes.

class FileOnDisk (*file_path, filename=None*)

Bases: `object`

Wrapper for a 'real' file on disk.

__init__ (*file_path, filename=None*)

Create a reference to a file on disk.

Parameters

- **file_path** (*str*) – File path or directory path
- **filename** (*str*) – If specified, `file_path` is considered to be a directory containing this filename. If not specified, the final element in `file_path` is considered the filename.

Raises `IOError` – if no such file exists

Examples

```
>>> a = FileOnDisk('/etc/resolv.conf')
>>> b = FileOnDisk('/etc', 'resolv.conf')
>>> a == b
True
```

add_to_archive (*tarf*)

Copy this file into the given tarfile object.

Parameters **tarf** (*tarfile.TarFile*) – Add this file to that archive.

close ()

Close the file previously opened.

copy_to (*dest_dir*)

Copy this file to the given destination directory.

Parameters **dest_dir** (*str*) – Destination directory or filename.

open (*mode*)

Open the file and return a reference to the file object.

Parameters **mode** (*str*) – Mode such as 'r', 'w', 'a', 'w+', etc.

Returns *file* – File object

exists

True if the file exists on disk, else False.

size

The size of this file, in bytes.

COT.platforms package reference

Package for identifying guest platforms and handling platform differences.

The *GenericPlatform* class describes the API and provides a generic implementation that can be overridden by subclasses to provide platform-specific logic.

In general, other modules should not instantiate subclasses directly but should instead use the *platform_from_product_class()* API to derive the appropriate subclass instance.

API

<i>is_known_product_class</i>	Determine if the given product class string is a known one.
<i>platform_from_product_class</i>	Get the class of Platform corresponding to a product class string.

Platform modules

<i>COT.platforms.generic</i>	API and generic implementation of platform-specific logic.
<i>COT.platforms.cisco_csr1000v</i>	Platform logic for the Cisco CSR1000V virtual router.
<i>COT.platforms.cisco_iosv</i>	Platform logic for the Cisco IOSv virtual router.
<i>COT.platforms.cisco_iosxrv</i>	Platform logic for the Cisco IOS XRV virtual router and its variants.
<i>COT.platforms.cisco_iosxrv_9000</i>	Platform logic for the Cisco IOS XRV 9000 virtual router.
<i>COT.platforms.cisco_nexus_9000v</i>	Platform logic for the Cisco Nexus 9000v virtual switch.
<i>COT.platforms.cisco_nxosv</i>	Platform logic for the Cisco NX-OSv virtual switch.

COT.platforms.generic module

API and generic implementation of platform-specific logic.

class **GenericPlatform**

Bases: *object*

Generic class for operations that depend on guest platform.

To be used whenever the guest is unrecognized or does not need special handling.

classmethod **controller_type_for_device** (*device_type*)

Get the default controller type for the given device type.

Parameters `device_type` (*str*) – ‘harddisk’, ‘cdrom’, etc.

Returns *str* – ‘ide’ unless overridden by subclass.

classmethod `guess_nic_name` (*nic_number*)

Guess the name of the Nth NIC for this platform.

Note: This method counts from 1, not from 0!

Parameters `nic_number` (*int*) – Nth NIC to name.

Returns *str* – “Ethernet1”, “Ethernet2”, etc. unless overridden by subclass.

classmethod `validate_cpu_count` (*cpus*)

Throw an error if the number of CPUs is not a supported value.

Parameters `cpus` (*int*) – Number of CPUs

Raises

- `ValueTooLowError` – if `cpus` is less than the minimum required by this platform
- `ValueTooHighError` – if `cpus` exceeds the maximum supported by this platform

classmethod `validate_memory_amount` (*mebibytes*)

Throw an error if the amount of RAM is not supported.

Parameters `mebibytes` (*int*) – RAM, in MiB.

Raises `ValueTooLowError` – if `mebibytes` is less than the minimum required by this platform

ValueTooHighError: if `mebibytes` is more than the maximum supported by this platform

classmethod `validate_nic_count` (*count*)

Throw an error if the number of NICs is not supported.

Parameters `count` (*int*) – Number of NICs.

Raises

- `ValueTooLowError` – if `count` is less than the minimum required by this platform
- `ValueTooHighError` – if `count` is more than the maximum supported by this platform

classmethod `validate_nic_type` (*type_string*)

Throw an error if the NIC type string is not supported.

See also:

- `COT.data_validation.canonicalize_nic_subtype()`
- `COT.data_validation.NIC_TYPES`

Parameters `type_string` (*str*) – See `COT.data_validation.NIC_TYPES`

Raises `ValueUnsupportedError` – if `type_string` is not in `SUPPORTED_NIC_TYPES`

classmethod `validate_nic_types` (*type_list*)

Throw an error if any NIC type string in the list is unsupported.

Parameters `type_list` (*list*) – See `COT.data_validation.NIC_TYPES`

Raises `ValueUnsupportedError` – if any value in `type_list` is not in `SUPPORTED_NIC_TYPES`

classmethod `validate_serial_count` (*count*)

Throw an error if the number of serial ports is not supported.

Parameters `count` (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if `count` is less than the minimum required by this platform
- `ValueTooHighError` – if `count` is more than the maximum supported by this platform

`BOOTSTRAP_DISK_TYPE = 'cdrom'`

`CONFIG_TEXT_FILE = 'config.txt'`

`LITERAL_CLI_STRING = 'config'`

`PLATFORM_NAME = '(unrecognized platform, generic)'`

`SECONDARY_CONFIG_TEXT_FILE = None`

`SUPPORTED_NIC_TYPES = ['E1000e', 'E1000', 'PCNet32', 'virtio', 'VMXNET3']`

`COT.platforms.cisco_csr1000v` module

Platform logic for the Cisco CSR1000V virtual router.

class `CSR1000V`

Bases: `COT.platforms.generic.GenericPlatform`

Platform-specific logic for Cisco CSR1000V platform.

classmethod `controller_type_for_device` (*device_type*)

CSR1000V uses SCSI for hard disks and IDE for CD-ROMs.

Parameters `device_type` (*str*) – ‘harddisk’ or ‘cdrom’

Returns *str* – ‘ide’ for CD-ROM, ‘scsi’ for hard disk

classmethod `guess_nic_name` (*nic_number*)

GigabitEthernet1, GigabitEthernet2, etc.

Warning: In all current CSR releases, NIC names start at “GigabitEthernet1”. Some early versions started at “GigabitEthernet0” but we don’t support that.

Parameters `nic_number` (*int*) – Nth NIC to name.

Returns

- “GigabitEthernet1”
- “GigabitEthernet2”

- etc.

classmethod `validate_cpu_count` (*cpus*)

CSR1000V supports 1, 2, 4, or 8 CPUs.

Parameters **`cpus`** (*int*) – Number of CPUs.

Raises

- `ValueTooLowError` – if `cpus` is less than 1
- `ValueTooHighError` – if `cpus` is more than 8
- `ValueUnsupportedError` – if `cpus` is an unsupported value between 1 and 8

classmethod `validate_memory_amount` (*mebibytes*)

Minimum 2.5 GiB, max 8 GiB.

Parameters **`mebibytes`** (*int*) – RAM, in MiB.

Raises

- `ValueTooLowError` – if `mebibytes` is less than 2560
- `ValueTooHighError` – if `mebibytes` is more than 8192

classmethod `validate_nic_count` (*count*)

CSR1000V requires 3 NICs and supports up to 26.

Parameters **`count`** (*int*) – Number of NICs.

Raises

- `ValueTooLowError` – if `count` is less than 3
- `ValueTooHighError` – if `count` is more than 26

classmethod `validate_serial_count` (*count*)

CSR1000V supports 0-2 serial ports.

Parameters **`count`** (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if `count` is less than 0
- `ValueTooHighError` – if `count` is more than 2

`CONFIG_TEXT_FILE` = 'iosxe_config.txt'

`LITERAL_CLI_STRING` = 'ios-config'

`PLATFORM_NAME` = 'Cisco CSR1000V'

`SUPPORTED_NIC_TYPES` = ['E1000', 'virtio', 'VMXNET3']

`COT.platforms.cisco_iosv` module

Platform logic for the Cisco IOSv virtual router.

class `IOSv`

Bases: `COT.platforms.generic.GenericPlatform`

Platform-specific logic for Cisco IOSv.

classmethod `guess_nic_name` (*nic_number*)

GigabitEthernet0/0, GigabitEthernet0/1, etc.

Parameters `nic_number` (*int*) – Nth NIC to name.

Returns

- “GigabitEthernet0/0”
- “GigabitEthernet0/1”
- etc.

classmethod `validate_cpu_count` (*cpus*)

IOSv only supports a single CPU.

Parameters `cpus` (*int*) – Number of CPUs.

Raises

- `ValueTooLowError` – if `cpus` is less than 1
- `ValueTooHighError` – if `cpus` is more than 1

classmethod `validate_memory_amount` (*mebibytes*)

IOSv has minimum 192 MiB (with minimal feature set), max 3 GiB.

Parameters `mebibytes` (*int*) – RAM, in MiB.

Raises

- `ValueTooLowError` – if `mebibytes` is less than 192
- `ValueTooHighError` – if `mebibytes` is more than 3072

classmethod `validate_nic_count` (*count*)

IOSv supports up to 16 NICs.

Parameters `count` (*int*) – Number of NICs.

Raises

- `ValueTooLowError` – if `count` is less than 0
- `ValueTooHighError` – if `count` is more than 16

classmethod `validate_serial_count` (*count*)

IOSv requires 1-2 serial ports.

Parameters `count` (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if `count` is less than 1
- `ValueTooHighError` – if `count` is more than 2

`BOOTSTRAP_DISK_TYPE = 'harddisk'`

`CONFIG_TEXT_FILE = 'ios_config.txt'`

`LITERAL_CLI_STRING = None`

`PLATFORM_NAME = 'Cisco IOSv'`

`SUPPORTED_NIC_TYPES = ['E1000']`

COT.platforms.cisco_iosxrv module

Platform logic for the Cisco IOS XRv virtual router and its variants.

Classes

<i>IOSXRv</i>	Platform-specific logic for Cisco IOS XRv platform.
<i>IOSXRvLC</i>	Platform-specific logic for Cisco IOS XRv line card.
<i>IOSXRvRP</i>	Platform-specific logic for Cisco IOS XRv HA-capable RP.

class IOSXRv

Bases: *COT.platforms.generic.GenericPlatform*

Platform-specific logic for Cisco IOS XRv platform.

classmethod guess_nic_name (*nic_number*)

MgmtEth0/0/CPU0/0, GigabitEthernet0/0/0/0, Gig0/0/0/1, etc.

Parameters *nic_number* (*int*) – Nth NIC to name.

Returns

- “MgmtEth0/0/CPU0/0”
- “GigabitEthernet0/0/0/0”
- “GigabitEthernet0/0/0/1”
- etc.

classmethod validate_cpu_count (*cpus*)

IOS XRv supports 1-8 CPUs.

Parameters *cpus* (*int*) – Number of CPUs

Raises

- *ValueTooLowError* – if *cpus* is less than 1
- *ValueTooHighError* – if *cpus* is more than 8

classmethod validate_memory_amount (*mebibytes*)

Minimum 3 GiB, max 8 GiB of RAM.

Parameters *mebibytes* (*int*) – RAM, in MiB.

Raises

- *ValueTooLowError* – if *mebibytes* is less than 3072
- *ValueTooHighError* – if *mebibytes* is more than 8192

classmethod validate_nic_count (*count*)

IOS XRv requires at least one NIC.

Parameters *count* (*int*) – Number of NICs.

Raises *ValueTooLowError* – if *count* is less than 1

classmethod validate_serial_count (*count*)

IOS XRv supports 1-4 serial ports.

Parameters *count* (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if count is less than 1
- `ValueTooHighError` – if count is more than 4

CONFIG_TEXT_FILE = 'iosxr_config.txt'

LITERAL_CLI_STRING = None

PLATFORM_NAME = 'Cisco IOS XRv'

SECONDARY_CONFIG_TEXT_FILE = 'iosxr_config_admin.txt'

SUPPORTED_NIC_TYPES = ['E1000', 'virtio']

class IOSXRvLC

Bases: `COT.platforms.cisco_iosxrv.IOSXRv`

Platform-specific logic for Cisco IOS XRv line card.

classmethod guess_nic_name (*nic_number*)

Fabric interface plus slot-appropriate GigabitEthernet interfaces.

Parameters *nic_number* (*int*) – Nth NIC to name.

Returns *str* – * “fabric” * “GigabitEthernet0/{SLOT}/0/0” * “GigabitEthernet0/{SLOT}/0/1” *
etc.

classmethod validate_serial_count (*count*)

No serial ports are needed but up to 4 can be used for debugging.

Parameters *count* (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if count is less than 0
- `ValueTooHighError` – if count is more than 4

CONFIG_TEXT_FILE = None

PLATFORM_NAME = 'Cisco IOS XRv line card'

SECONDARY_CONFIG_TEXT_FILE = None

class IOSXRvRP

Bases: `COT.platforms.cisco_iosxrv.IOSXRv`

Platform-specific logic for Cisco IOS XRv HA-capable RP.

classmethod guess_nic_name (*nic_number*)

Fabric and management only.

Parameters *nic_number* (*int*) – Nth NIC to name.

Returns *str* – “fabric” or “MgmtEth0/{SLOT}/CPU0/0” only

classmethod validate_nic_count (*count*)

Fabric plus an optional management NIC.

Parameters *count* (*int*) – Number of NICs.

Raises

- `ValueTooLowError` – if count is less than 1
- `ValueTooHighError` – if count is more than 2

PLATFORM_NAME = 'Cisco IOS XRv route processor card'

COT.platforms.cisco_iosxrv_9000 module

Platform logic for the Cisco IOS XRv 9000 virtual router.

class IOSXRv9000

Bases: `COT.platforms.cisco_iosxrv.IOSXRv`

Platform-specific logic for Cisco IOS XRv 9000 platform.

classmethod guess_nic_name (*nic_number*)

MgmtEth0/0/CPU0/0, CtrlEth, DevEth, GigabitEthernet0/0/0/0, etc.

Parameters *nic_number* (*int*) – Nth NIC to name.

Returns

- “MgmtEth0/0/CPU0/0”
- “CtrlEth”
- “DevEth”
- “GigabitEthernet0/0/0/0”
- “GigabitEthernet0/0/0/1”
- etc.

classmethod validate_cpu_count (*cpus*)

Minimum 1, maximum 32 CPUs.

Parameters *cpus* (*int*) – Number of CPUs

Raises

- `ValueTooLowError` – if *cpus* is less than 1
- `ValueTooHighError` – if *cpus* is more than 32

classmethod validate_memory_amount (*mebibytes*)

Minimum 8 GiB, no known maximum (128GiB+ is permitted).

Parameters *mebibytes* (*int*) – RAM, in MiB.

Raises `ValueTooLowError` – if *mebibytes* is less than 8192

classmethod validate_nic_count (*count*)

IOS XRv 9000 requires at least 4 NICs.

Parameters *count* (*int*) – Number of NICs.

Raises `ValueTooLowError` – if *count* is less than 4

PLATFORM_NAME = ‘Cisco IOS XRv 9000’

SUPPORTED_NIC_TYPES = [‘E1000’, ‘virtio’, ‘VMXNET3’]

COT.platforms.cisco_nexus_9000v module

Platform logic for the Cisco Nexus 9000v virtual switch.

class Nexus9000v

Bases: `COT.platforms.generic.GenericPlatform`

Platform-specific logic for Cisco Nexus 9000v.

classmethod `guess_nic_name` (*nic_number*)

The Nexus 9000v has a management NIC and some number of data NICs.

Parameters `nic_number` (*int*) – Nth NIC to name.

Returns

- `mgmt0`
- `Ethernet1/1`
- `Ethernet1/2`
- ...

classmethod `validate_cpu_count` (*cpus*)

The Nexus 9000v requires 1-4 vCPUs.

Parameters `cpus` (*int*) – Number of vCPUs

Raises

- `ValueTooLowError` – if `cpus` is less than 1
- `ValueTooHighError` – if `cpus` is more than 4

classmethod `validate_memory_amount` (*mebibytes*)

The Nexus 9000v requires at least 8 GiB of RAM.

Parameters `mebibytes` (*int*) – RAM, in MiB.

Raises `ValueTooLowError` – if `mebibytes` is less than 8192

classmethod `validate_nic_count` (*count*)

The Nexus 9000v requires at least 1 and supports at most 65 NICs.

Parameters `count` (*int*) – Number of NICs.

Raises

- `ValueTooLowError` – if `count` is less than 1
- `ValueTooHighError` – if `count` is more than 65

classmethod `validate_serial_count` (*count*)

The Nexus 9000v requires exactly 1 serial port.

Parameters `count` (*int*) – Number of serial ports.

Raises

- `ValueTooLowError` – if `count` is less than 1
- `ValueTooHighError` – if `count` is more than 1

`CONFIG_TEXT_FILE = 'nxos_config.txt'`

`LITERAL_CLI_STRING = None`

`PLATFORM_NAME = 'Cisco Nexus 9000v'`

`SUPPORTED_NIC_TYPES = ['E1000', 'VMXNET3']`

COT.platforms.cisco_nxosv module

Platform logic for the Cisco NX-OSv virtual switch.

class NXOSv

Bases: *COT.platforms.generic.GenericPlatform*

Platform-specific logic for Cisco NX-OSv (Titanium).

classmethod guess_nic_name (*nic_number*)

NX-OSv names its NICs a bit interestingly...

Parameters *nic_number* (*int*) – Nth NIC to name.

Returns

- mgmt0
- Ethernet2/1
- Ethernet2/2
- ...
- Ethernet2/48
- Ethernet3/1
- Ethernet3/2
- ...

classmethod validate_cpu_count (*cpus*)

NX-OSv requires 1-8 CPUs.

Parameters *cpus* (*int*) – Number of CPUs

Raises

- *ValueTooLowError* – if *cpus* is less than 1
- *ValueTooHighError* – if *cpus* is more than 8

classmethod validate_memory_amount (*mebibytes*)

NX-OSv requires 2-8 GiB of RAM.

Parameters *mebibytes* (*int*) – RAM, in MiB.

Raises *ValueTooLowError* – if *mebibytes* is less than 2048 *ValueTooHighError*: if *mebibytes* is more than 8192

classmethod validate_serial_count (*count*)

NX-OSv requires 1-2 serial ports.

Parameters *count* (*int*) – Number of serial ports.

Raises

- *ValueTooLowError* – if *count* is less than 1
- *ValueTooHighError* – if *count* is more than 2

CONFIG_TEXT_FILE = 'nxos_config.txt'

LITERAL_CLI_STRING = None

PLATFORM_NAME = 'Cisco NX-OSv'

SUPPORTED_NIC_TYPES = ['E1000', 'virtio']

is_known_product_class (*product_class*)

Determine if the given product class string is a known one.

Parameters **product_class** (*str*) – String such as ‘com.cisco.iosv’

Returns *bool* – Whether product_class is known.

Examples

```
>>> is_known_product_class("com.cisco.n9k")
True
>>> is_known_product_class("foobar")
False
```

platform_from_product_class (*product_class*)

Get the class of Platform corresponding to a product class string.

Parameters **product_class** (*str*) – String such as ‘com.cisco.iosv’

Returns *class* – GenericPlatform or a subclass of it

Examples

```
>>> platform_from_product_class("com.cisco.n9k")
<class 'COT.platforms.cisco_nexus_9000v.Nexus9000v'>
>>> platform_from_product_class(None)
<class 'COT.platforms.generic.GenericPlatform'>
>>> platform_from_product_class("frobozz")
<class 'COT.platforms.generic.GenericPlatform'>
```

User interface modules

<i>COT.ui_shared</i>	Abstract user interface superclass.
<i>COT.cli</i>	CLI entry point for the Common OVF Tool (COT) suite.

COT.ui_shared module

Abstract user interface superclass.

class UI (*force=False*)

Bases: *object*

Abstract user interface functionality.

Can also be used in test code as a stub that autoconfirms everything.

__init__ (*force=False*)

Constructor.

Parameters **force** (*bool*) – See *force*.

choose_from_list (*footer, option_list, default_value, header='', info_list=None*)

Prompt the user to choose from a list.

Parameters

- **footer** (*str*) – Prompt string to display following the list
- **option_list** (*list*) – List of strings to choose amongst
- **default_value** (*str*) – Default value to select if user declines
- **header** (*str*) – String to display prior to the list
- **info_list** (*list*) – Verbose strings to display in place of *option_list*

Returns *str* – *default_value* or an item from *option_list*.

confirm (*prompt*)

Prompt user to confirm the requested operation.

Auto-accepts if *force* is set to True.

Warning: This stub implementation does not actually interact with the user, but instead returns *default_confirm_response*. Subclasses should override this method.

Parameters **prompt** (*str*) – Message to prompt the user with

Returns *bool* – True (user confirms acceptance) or False (user declines)

confirm_or_die (*prompt*)

If the user doesn't agree, abort the program.

A simple wrapper for *confirm()* that calls *sys.exit()* if *confirm()* returns False.

Parameters **prompt** (*str*) – Message to prompt the user with

Raises *SystemExit* – if user declines

fill_examples (*example_list*)

Pretty-print a set of usage examples.

Parameters **example_list** (*list*) – List of (example, description) tuples.

Raises *NotImplementedError* – Must be implemented by a subclass.

fill_usage (*subcommand*, *usage_list*)

Pretty-print a list of usage strings.

Parameters

- **subcommand** (*str*) – Subcommand name/keyword
- **usage_list** (*list*) – List of usage strings for this subcommand.

Returns *str* – Concatenation of all usage strings, each appropriately wrapped to the *terminal_width* value.

get_input (*prompt*, *default_value*)

Prompt the user to enter a string.

Auto-inputs the *default_value* if *force* is set to True.

Warning: This stub implementation does not actually interact with the user, but instead always returns *default_value*. Subclasses should override this method.

Parameters

- **prompt** (*str*) – Message to prompt the user with
- **default_value** (*str*) – Default value to input if the user simply hits Enter without entering a value, or if *force*.

Returns *str* – Input value

get_password (*username*, *host*)

Get password string from the user.

Parameters

- **username** (*str*) – Username the password is associated with
- **host** (*str*) – Host the password is associated with

Raises `NotImplementedError` – Must be implemented by a subclass.

default_confirm_response = `None`

Knob for API testing, sets the default response to confirm().

force = `None`

Whether to automatically select the default value in all cases.

(As opposed to interactively prompting the user.)

terminal_width

Get the width of the terminal in columns.

COT.cli module

CLI entry point for the Common OVF Tool (COT) suite.

Functions

formatter Create formatter for log output.

Classes

CLI Command-line user interface for COT.

class CLI (*terminal_width=None*)

Bases: `COT.ui_shared.UI`

Command-line user interface for COT.

<i>confirm</i>	Prompt user to confirm the requested operation.
<i>create_parser</i>	Create parser object for global cot command.
<i>create_subparsers</i>	Populate the CLI sub-parsers for all known submodules.
<i>fill_examples</i>	Pretty-print a set of usage examples.
<i>fill_usage</i>	Pretty-print a list of usage strings for a COT subcommand.
<i>get_input</i>	Prompt the user to enter a string.
<i>get_password</i>	Get password string from the user.
<i>main</i>	Main worker function for COT when invoked from the CLI.
<i>parse_args</i>	Parse the given CLI arguments into a namespace object.
<i>run</i>	Parse the given CLI args then run.
<i>set_verbosity</i>	Enable logging and/or change the logging verbosity level.
<i>terminal_width</i>	The width of the terminal in columns.

__init__ (*terminal_width=None*)

Create CLI handler instance.

Parameters **terminal_width** (*int*) – (optional) Set the terminal width for this CLI, independent of the actual terminal in use.

add_subparser (*title*, *parent=None*, *aliases=None*, *lookup_prefix=''*, ***kwargs*)

Create a subparser under the specified parent.

Parameters

- **title** (*str*) – Canonical keyword for this subparser
- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **aliases** (*list*) – Aliases for `title`. Only used in Python 3.x.
- **lookup_prefix** (*str*) – String to prepend to `title` and each alias in `aliases` for lookup purposes.
- **kwargs** (*dict*) – Passed through to `parent.add_parser()`

Returns *object* – Subparser object

static args_to_dict (*args*)

Convert args to a dict and perform any needed cleanup.

Parameters **args** (*argparse.Namespace*) – Namespace from `parse_args()`.

Returns *dict* – Dictionary of arg to value

confirm (*prompt*)

Prompt user to confirm the requested operation.

Auto-accepts if `force` is set to `True`.

Parameters **prompt** (*str*) – Message to prompt the user with

Returns *bool* – `True` (user accepts) or `False` (user declines)

create_parser ()

Create parser object for global `cot` command.

Includes a number of globally applicable CLI options.

create_subparsers ()

Populate the CLI sub-parsers for all known submodules.

Creates an instance of each `COTGenericSubmodule` subclass, then calls `create_subparser()` for each.

fill_examples (*example_list*)

Pretty-print a set of usage examples.

Parameters **example_list** (*list*) – List of (description, CLI example) tuples.

Returns *str* – Concatenation of examples, each wrapped appropriately to the `terminal_width()` value. CLI examples will be wrapped with backslashes and a hanging indent.

Examples

```
>>> print(CLI(68).fill_examples([
...     ("Deploy to vSphere/ESXi server 192.0.2.100 with credentials"
...      " admin/admin, creating a VM named 'test_vm' from foo.ova.",
...      'cot deploy foo.ova esxi 192.0.2.100 -u admin -p admin'
```

```
...     '-n test_vm'),
...     ("Deploy to vSphere/ESXi server 192.0.2.100, with username"
...      " admin (prompting the user to input a password at runtime)",
...      " creating a VM based on profile '1CPU-2.5GB' in foo.ova.",
...      'cot deploy foo.ova esxi 192.0.2.100 -u admin -c 1CPU-2.5GB')
... ]))
```

Examples:

Deploy to vSphere/ESXi server 192.0.2.100 with credentials admin/admin, creating a VM named 'test_vm' from foo.ova.

```
cot deploy foo.ova esxi 192.0.2.100 -u admin -p admin \
    -n test_vm
```

Deploy to vSphere/ESXi server 192.0.2.100, with username admin (prompting the user to input a password at runtime), creating a VM based on profile '1CPU-2.5GB' in foo.ova.

```
cot deploy foo.ova esxi 192.0.2.100 -u admin -c 1CPU-2.5GB
```

fill_usage (*subcommand*, *usage_list*)

Pretty-print a list of usage strings for a COT subcommand.

Automatically prepends a `cot subcommand --help` usage string to the provided list.

Parameters

- **subcommand** (*str*) – Subcommand name/keyword
- **usage_list** (*list*) – List of usage strings for this subcommand.

Returns *string* – All usage strings, each appropriately wrapped to the `terminal_width()` value.

Examples

```
>>> print (CLI(50).fill_usage('add-file',
...     ["FILE PACKAGE [-o OUTPUT] [-f FILE_ID]"]))

cot add-file --help
cot <opts> add-file FILE PACKAGE [-o OUTPUT]
                        [-f FILE_ID]
```

get_input (*prompt*, *default_value*)

Prompt the user to enter a string.

Auto-inputs the `default_value` if `force` is set to `True`.

Parameters

- **prompt** (*str*) – Message to prompt the user with
- **default_value** (*str*) – Default value to input if the user simply hits Enter without entering a value, or if `force`.

Returns *str* – Input value

get_password (*username*, *host*)

Get password string from the user.

Parameters

- **username** (*str*) – Username the password is associated with
- **host** (*str*) – Host the password is associated with

Raises `InvalidInputError` – if `force` is `True` (as there is no “default” password value)

Returns *str* – Password string

main (*args*)

Main worker function for COT when invoked from the CLI.

- Calls `set_verbosity()` with the appropriate verbosity level derived from the args.
- Looks up the appropriate `COTGenericSubmodule` instance corresponding to the subcommand that was invoked.
- Converts `args` to a dict and calls `set_value()` for each arg/value in the dict.
- Calls `run()` followed by `finished()`.
- Catches various exceptions and handles them appropriately.

Parameters **args** (*argparse.Namespace*) – Parser namespace object returned from `parse_args()`.

Returns

int – Exit code for the COT executable.

- 0 on successful completion
- 1 on runtime error
- 2 on input error (parser error, `InvalidInputError`, etc.)

parse_args (*argv*)

Parse the given CLI arguments into a namespace object.

Parameters **argv** (*list*) – List of CLI arguments, not including `argv0`

Returns *argparse.Namespace* – Parser namespace object

run (*argv*)

Parse the given CLI args then run.

Calls `parse_args()` followed by `main()`.

Parameters **argv** (*list*) – The CLI argv value (not including `argv[0]`)

Returns *int* – Return code from `main()`

static set_instance_attributes (*arg_dict*)

Set attributes of the instance based on the given `arg_dict`.

Parameters **arg_dict** (*dict*) – Dictionary of (attribute, value).

Raises `InvalidInputError` – if attributes are not validly set.

set_verbosity (*level*)

Enable logging and/or change the logging verbosity level.

Will call `formatter()` and associate the resulting formatter with logging.

Parameters **level** (*int*) – Logging level as defined by `logging`

terminal_width

The width of the terminal in columns.

formatter (*verbosity=20*)

Create formatter for log output.

We offer different (more verbose) formatting when debugging is enabled, hence this need.

Parameters **verbosity** (*int*) – Logging level as defined by `logging`.

Returns `colorlog.ColoredFormatter` – Formatter object to use with `logging`.

main ()

Launch COT from the CLI.

Sub-packages

<code>COT.helpers</code>	Provides a common interface for interacting with various non-Python programs.
<code>COT.ovf</code>	Package for handling OVF and OVA virtual machine description files.

COT.helpers package reference

Provides a common interface for interacting with various non-Python programs.

API

<code>helpers</code>	Dictionary of concrete Helper subclasses to be populated at load time.
<code>helper_select</code>	Select the first helper that is available from the given list.

Exceptions

<code>HelperError</code>	A helper program exited with non-zero return code.
<code>HelperNotFoundError</code>	A helper program cannot be located.

Helper modules

<code>COT.helpers.helper</code>	Common interface for providers of non-Python helper programs.
<code>COT.helpers.apt_get</code>	Wrapper for the ‘apt-get’ package manager.
<code>COT.helpers.brew</code>	Wrapper for the Homebrew ‘brew’ package manager for Mac (http://brew.sh).
<code>COT.helpers.fatdisk</code>	Give COT access to <code>fatdisk</code> for creating and updating FAT32 file systems.
<code>COT.helpers.gcc</code>	Give COT access to <code>gcc</code> command for building other helpers.
<code>COT.helpers.isoinfo</code>	Give COT access to <code>isoinfo</code> for inspecting ISO images.
<code>COT.helpers.make</code>	Give COT access to <code>make</code> command for building other helpers.
<code>COT.helpers.mkisofs</code>	Give COT access to <code>mkisofs</code> , <code>genisoimage</code> , or <code>xorriso</code> for creating ISO images.
<code>COT.helpers.ovftool</code>	Give COT access to <code>ovftool</code> for validating and deploying OVF to ESXi.
<code>COT.helpers.port</code>	Wrapper for the MacPorts ‘port’ package manager.
<code>COT.helpers.qemu_img</code>	Give COT access to <code>qemu-img</code> for manipulating disk image formats.
<code>COT.helpers.vmdktool</code>	Give COT access to <code>vmdktool</code> for manipulating compressed VMDK files.
<code>COT.helpers.yum</code>	Wrapper for the ‘yum’ package manager.

COT.helpers.helper module

Common interface for providers of non-Python helper programs.

Provides the ability to install the program if not already present, and the ability to run the program as well.

Classes

<i>HelperNotFoundError</i>	A helper program cannot be located.
<i>HelperError</i>	A helper program exited with non-zero return code.
<i>Helper</i>	A provider of a non-Python helper program.
<i>PackageManager</i>	Helper program with additional API method <code>install_package()</code> .

Attributes

<i>helpers</i>	Dictionary of concrete Helper subclasses to be populated at load time.
----------------	--

Functions

<i>check_call</i>	Wrapper for <code>subprocess.check_call()</code> .
<i>check_output</i>	Wrapper for <code>subprocess.check_output()</code> .

exception HelperError

Bases: `exceptions.EnvironmentError`

A helper program exited with non-zero return code.

exception HelperNotFoundError

Bases: `exceptions.OSError`

A helper program cannot be located.

class Helper (*name*, *info_uri*=None, *version_args*=None, *version_regex*='([0-9.]+'')

Bases: `object`

A provider of a non-Python helper program.

Static Methods

<i>cp</i>	Copy the given src to the given dest, using sudo if needed.
<i>download_and_expand_tgz</i>	Context manager for downloading and expanding a .tar.gz file.
<i>mkdir</i>	Check whether the given target directory exists, and create if not.

Instance Properties

<i>name</i>	Name of the helper program.
<i>info_uri</i>	URI for more information about this helper.
<i>installable</i>	Whether COT is capable of installing this program on this system.
<i>installed</i>	Whether this helper program is installed and available to run.
<i>path</i>	Discovered path to the helper.
<i>version</i>	Release version of the associated helper program.

Instance Methods

<code>call</code>	Call the helper program with the given arguments.
<code>install</code>	Install the helper program.
<code>_install</code>	Subclass-specific implementation of installation logic.
<code>unsure_how_to_install</code>	Return a <code>RuntimeError</code> or <code>NotImplementedError</code> for install trouble.

`__init__` (*name*, *info_uri*=None, *version_args*=None, *version_regexp*='([0-9.]+')')
 Initializer.

Parameters

- **name** (*str*) – Name of helper executable
- **info_uri** (*str*) – URI to refer to for more info about this helper.
- **version_args** (*list*) – Args to pass to the helper to get its version. Defaults to `['--version']` if unset.
- **version_regexp** (*str*) – Regexp to get the version number from the output of the command.

`_install` ()
 Subclass-specific implementation of installation logic.

This method should only be called from `install()`, which does the appropriate pre-validation against the `installed` and `installable` properties before calling into this method if appropriate.

`call` (*args*, *capture_output*=True, ***kwargs*)
 Call the helper program with the given arguments.

Parameters

- **args** (*list*) – List of arguments to the helper program.
- **capture_output** (*boolean*) – If True, stdout/stderr will be redirected to a buffer and returned, instead of being displayed to the user. (I.e., `check_output()` will be invoked instead of `check_call()`)

Returns *str* – Captured stdout/stderr if `capture_output` is True, else None.

For the other parameters, see `check_call()` and `check_output()`.

Raises `HelperNotFoundError` – if the helper was not previously installed, and the user declines to install it at this time.

`static cp` (*src*, *dest*)
 Copy the given *src* to the given *dest*, using `sudo` if needed.

Parameters

- **src** (*str*) – Source path.
- **dest** (*str*) – Destination path.

Returns *bool* – True

Raises `HelperError` – if file copying fails

`static download_and_expand_tgz` (**args*, ***kws*)
 Context manager for downloading and expanding a .tar.gz file.

Creates a temporary directory, downloads the specified URL into the directory, unzips and untars the file into this directory, then yields to the given block. When the block exits, the temporary directory and its contents are deleted.

```
with download_and_expand_tgz("http://example.com/foo.tgz") as d:
    # archive contents have been extracted to 'd'
    ...
# d is automatically cleaned up.
```

Parameters `url` (*str*) – URL of a .tgz or .tar.gz file to download.

Yields *str* – Temporary directory path where the archive has been extracted.

install()

Install the helper program.

Raises

- `NotImplementedError` – if not *installable* on this platform
- `RuntimeError` – if potentially *installable* on this platform but required helpers (e.g., package managers) are not available.
- `HelperError` – if installation is attempted but fails.

Subclasses should not override this method but instead should provide an appropriate implementation of the `_install()` method.

static mkdir (*directory*, *permissions=493*)

Check whether the given target directory exists, and create if not.

Parameters

- **directory** (*str*) – Directory to check/create.
- **permissions** (*int*) – Permission mask to set when creating a directory. Default is 0o755.

unsure_how_to_install()

Return a `RuntimeError` or `NotImplementedError` for install trouble.

USER_INTERFACE = <COT.cli.CLI object>

User interface (if any) available to helpers.

_provider_package = {}

Mapping of package manager name to package name to install with it.

info_uri

URI for more information about this helper.

installable

Whether COT is capable of installing this program on this system.

installed

Whether this helper program is installed and available to run.

name

Name of the helper program.

path

Discovered path to the helper.

version

Release version of the associated helper program.

class HelperDict (*factory*, **args*, ***kwargs*)

Bases: `dict`

Dictionary of Helper objects by name.

Similar to `collections.defaultdict` but takes the key as a parameter to the factory.

__init__ (*factory*, *args, **kwargs)

Create the given dictionary with the given factory class/method.

Parameters **factory** (*object*) – Factory class or method to be called to populate a new entry in response to `__missing__()`.

For the other parameters, see `dict`.

class PackageManager (*name*, *info_uri=None*, *version_args=None*, *version_regexp='([0-9.]*)'*)

Bases: `COT.helpers.helper.Helper`

Helper program with additional API method `install_package()`.

install_package (*package*)

Install the requested package if needed.

Parameters **package** (*str*) – Name of the package to install, or a list of parameters used to install the package.

check_call (*args*, *require_success=True*, *retry_with_sudo=False*, **kwargs)

Wrapper for `subprocess.check_call()`.

Unlike `check_output()` below, this does not redirect stdout or stderr; all output from the subprocess will be sent to the system stdout/stderr as normal.

Parameters

- **args** (*list*) – Command to invoke and its associated args
- **require_success** (*boolean*) – If `False`, do not raise an error when the command exits with a return code other than 0
- **retry_with_sudo** (*boolean*) – If `True`, if the command gets an exception, prepend `sudo` to the command and try again.

For the other parameters, see `subprocess.check_call()`.

Raises

- `HelperNotFoundError` – if the command doesn't exist (instead of a `OSError`)
- `HelperError` – if `require_success` is not `False` and the command returns a value other than 0 (instead of a `subprocess.CalledProcessError`).
- `OSError` – as `subprocess.check_call()`.

Examples

```
>>> check_call(['true'])
>>> try:
...     check_call(['false'])
... except HelperError as e:
...     print(e.errno)
...     print(e.strerror)
1
Helper program 'false' exited with error 1
>>> check_call(['false'], require_success=False)
>>> try:
...     check_call(['/non/exist'])
```

```
... except HelperNotFoundError as e:
...     print(e.errno)
...     print(e.strerror)
2
Unable to locate helper program '/non/exist'. Please check your $PATH.
>>> try:
...     check_call(['/etc/'])
... except OSError as e:
...     print(e.errno)
...     print(e.strerror)
13
Permission denied
```

check_output (*args*, *require_success=True*, *retry_with_sudo=False*, ***kwargs*)

Wrapper for `subprocess.check_output()`.

Automatically redirects stderr to stdout, captures both to a buffer, and generates a debug message with the stdout contents.

Parameters

- **args** (*list*) – Command to invoke and its associated args
- **require_success** (*boolean*) – If False, do not raise an error when the command exits with a return code other than 0
- **retry_with_sudo** (*boolean*) – If True, if the command gets an exception, prepend sudo to the command and try again.

For the other parameters, see `subprocess.check_output()`.

Returns *str* – Captured stdout/stderr from the command

Raises

- *HelperNotFoundError* – if the command doesn't exist (instead of a *OSError*)
- *HelperError* – if `require_success` is not False and the command returns a value other than 0 (instead of a `subprocess.CalledProcessError`).
- *OSError* – as `subprocess.check_output()`.

Examples

```
>>> output = check_output(['echo', 'Hello world!'])
>>> assert output == "Hello world!\n"
>>> try:
...     check_output(['false'])
... except HelperError as e:
...     print(e.errno)
...     print(e.strerror)
1
Helper program 'false' exited with error 1:
> false

>>> output = check_output(['false'], require_success=False)
>>> assert output == ''
>>> try:
...     check_output(['/non/exist'])
... except HelperNotFoundError as e:
```

```

...     print(e.errno)
...     print(e.strerror)
2
Unable to locate helper program '/non/exist'. Please check your $PATH.
>>> try:
...     check_output(['/etc/'])
... except OSError as e:
...     print(e.errno)
...     print(e.strerror)
13
Permission denied

```

helper_select (*choices*)

Select the first helper that is available from the given list.

If no helper in the list is currently installed, will install the first installable helper from the list.

Raises *HelperNotFoundError* – if no valid helper is available or installable.

Parameters *choices* (*list*) – List of helpers, in order from most preferred to least preferred.
Each choice in this list can be either:

- a string (the helper name, such as “mkisofs”)
- a tuple of (name, minimum version) such as (“qemu-img”, “2.1.0”).

Returns *Helper* – The selected helper class instance.

helpers = {'isoinfo': <COT.helpers.isoinfo.ISOInfo object at 0x7f25e4b046d0>, 'gcc': <COT.helpers.gcc.GCC object at 0x7f25e4b046d0>}

Dictionary of concrete Helper subclasses to be populated at load time.

COT.helpers.apt_get module

Wrapper for the ‘apt-get’ package manager.

class AptGet

Bases: *COT.helpers.helper.PackageManager*

The ‘apt-get’ package manager utility.

__init__ ()

Initializer.

install_package (*package*)

Install the requested package if needed.

Parameters *package* (*str*) – Name of the package to install.

COT.helpers.brew module

Wrapper for the Homebrew ‘brew’ package manager for Mac (<http://brew.sh>).

class Brew

Bases: *COT.helpers.helper.PackageManager*

The ‘brew’ package manager utility.

__init__ ()

Initializer.

install_package (*package*)

Install the requested package if needed.

Parameters **package** (*str*) – Name of the package to install, or a list of parameters used to install the package.

COT.helpers.fatdisk module

Give COT access to `fatdisk` for creating and updating FAT32 file systems.

<http://github.com/goblinhack/fatdisk>

class FatDisk

Bases: *COT.helpers.helper.Helper*

Wrapper for `fatdisk` (<http://github.com/goblinhack/fatdisk>).

__init__ ()
Initializer.

installable
Whether COT is capable of installing this program on this system.

COT.helpers.gcc module

Give COT access to `gcc` command for building other helpers.

class GCC

Bases: *COT.helpers.helper.Helper*

Helper provider for `gcc` command.

__init__ ()
Initializer.

COT.helpers.isoinfo module

Give COT access to `isoinfo` for inspecting ISO images.

<http://cdrecord.org/>

class ISOInfo

Bases: *COT.helpers.helper.Helper*

Helper provider for `isoinfo`.

<http://cdrecord.org/>

__init__ ()
Initializer.

COT.helpers.make module

Give COT access to `make` command for building other helpers.

class Make

Bases: *COT.helpers.helper.Helper*

Helper provider for make command.

`__init__()`
Initializer.

COT.helpers.mkisofs module

Give COT access to mkisofs, genisoimage, or xorriso for creating ISO images.

<http://cdrecord.org/> <https://www.gnu.org/software/xorriso/>

Classes

<i>MkISOFS</i>	Helper provider for mkisofs.
<i>GenISOImage</i>	Helper provider for genisoimage, a fork of mkisofs.
<i>XorrISO</i>	Helper provider for xorriso.

class GenISOImage

Bases: *COT.helpers.helper.Helper*

Helper provider for genisoimage, a fork of mkisofs.

`__init__()`
Initializer.

class MkISOFS

Bases: *COT.helpers.helper.Helper*

Helper provider for mkisofs.

<http://cdrecord.org/>

`__init__()`
Initializer.

class XorrISO

Bases: *COT.helpers.helper.Helper*

Helper provider for xorriso.

<https://www.gnu.org/software/xorriso/>

`__init__()`
Initializer.

COT.helpers.ovftool module

Give COT access to ovftool for validating and deploying OVF to ESXi.

<https://www.vmware.com/support/developer/ovf/>

class OVFTool

Bases: *COT.helpers.helper.Helper*

Helper provider for ovftool from VMware.

<https://www.vmware.com/support/developer/ovf/>

__init__()
Initializer.

unsure_how_to_install()
Return a NotImplementedError about missing install logic.

We override the default install implementation to raise a more detailed error message for ovftool.

installable
COT can't install ovftool because of VMware site restrictions.

COT.helpers.port module

Wrapper for the MacPorts 'port' package manager.

class Port

Bases: *COT.helpers.helper.PackageManager*

The 'port' package manager utility.

__init__()
Initializer.

install_package(*package*)
Install the requested package if needed.

Parameters **package** (*str*) – Name of the package to install.

COT.helpers.qemu_img module

Give COT access to qemu-img for manipulating disk image formats.

<http://www.qemu.org>

class QEMUImg

Bases: *COT.helpers.helper.Helper*

Helper provider for qemu-img (<http://www.qemu.org>).

__init__()
Initializer.

COT.helpers.vmdktool module

Give COT access to vmdktool for manipulating compressed VMDK files.

<http://www.freshports.org/sysutils/vmdktool/>

class VMDKTool

Bases: *COT.helpers.helper.Helper*

Helper provider for vmdktool.

<http://www.freshports.org/sysutils/vmdktool/>

__init__()
Initializer.

installable
Whether COT is capable of installing this program on this system.

COT.helpers.yum module

Wrapper for the ‘yum’ package manager.

class Yum

Bases: *COT.helpers.helper.PackageManager*

The ‘yum’ package manager utility.

__init__()
Initializer.

install_package(*package*)
Install the requested package if needed.

Parameters **package** (*str*) – Name of the package to install.

COT.ovf package reference

Package for handling OVF and OVA virtual machine description files.

The `COT.ovf.OVF` class provides an implementation of the *COT.vm_description.VMDescription* interface. In general, COT submodules should be agnostic of the internals of this package and should only use the `VMDescription` interface.

API

<i>OVF</i>	Representation of the contents of an OVF or OVA.
------------	--

Exceptions

<i>OVFHardwareDataError</i>	The input data used to construct an <code>OVFHardware</code> is not sane.
<i>OVFItemDataError</i>	Data to be added to an <code>OVFItem</code> conflicts with existing data.

Modules

<i>COT.ovf.ovf</i>	Module for handling OVF and OVA virtual machine description files.
<i>COT.ovf.hardware</i>	Representation of OVF hardware definitions.
<i>COT.ovf.item</i>	Module for working with individual hardware elements in an OVF.
<i>COT.ovf.name_helper</i>	Module for handling the differences in XML between OVF spec versions.

COT.ovf.ovf module

Module for handling OVF and OVA virtual machine description files.

Functions

<i>byte_count</i>	Convert an OVF-style value + multiplier into decimal byte count.
<i>byte_string</i>	Pretty-print the given bytes value.
<i>factor_bytes</i>	Convert a byte count into OVF-style bytes + multiplier.

Classes

<i>OVF</i>	Representation of the contents of an OVF or OVA.
------------	--

class **OVF** (*input_file*, *output_file*)

Bases: *COT.vm_description.VMDescription*, *COT.xml_file.XML*

Representation of the contents of an OVF or OVA.

Properties

<i>input_file</i>	Data file to read in.
<i>output_file</i>	OVF or OVA file that will be created or updated by <i>write()</i> .
<i>ovf_version</i>	Float representing the OVF specification version in use.
<i>product_class</i>	The product class identifier, such as com.cisco.csr1000v.
<i>platform</i>	The platform type, as determined from the OVF descriptor.
<i>config_profiles</i>	The list of supported configuration profiles.
<i>default_config_profile</i>	The name of the default configuration profile.
<i>environment_properties</i>	The array of environment properties.
<i>environment_transports</i>	The list of environment transport method strings.
<i>networks</i>	The list of network names currently defined in this VM.
<i>network_descriptions</i>	The list of network descriptions currently defined in this VM.
<i>system_types</i>	List of virtual system type(s) supported by this virtual machine.
<i>version_short</i>	Short descriptive version string (XML <i>Version</i> element).
<i>version_long</i>	Long descriptive version string (XML <i>FullVersion</i> element).

__init__ (*input_file*, *output_file*)

Open the specified OVF and read its XML into memory.

Parameters

- **input_file** (*str*) – Data file to read in.
- **output_file** (*str*) – File name to write to. If this VM is read-only, (there will never be an output file) this value should be *None*; if the output filename is not yet known, use "" and subsequently set *output_file* when it is determined.

Raises

- *VMInitError*
 - if the OVF descriptor cannot be located
 - if an XML parsing error occurs
 - if the XML is not actually an OVF descriptor
 - if the OVF hardware validation fails
- *Exception* – will call *destroy()* to clean up before reraising any exception encountered.

add_controller_device (*device_type*, *subtype*, *address*, *ctrl_item=None*)

Create a new IDE or SCSI controller, or update existing one.

Parameters

- **device_type** (*str*) – 'ide' or 'scsi'

- **subtype** (*object*) – (Optional) subtype string such as 'virtio' or list of subtype strings
- **address** (*int*) – Controller address such as 0 or 1 (optional)
- **ctrl_item** (*OVFItem*) – Existing controller device to update (optional)

Returns *OVFItem* – New or updated controller device object

Raises *ValueTooHighError* – if no more controllers can be created

add_disk (*disk_repr, file_id, drive_type, disk=None*)

Add a new disk object to the VM or overwrite the provided one.

Parameters

- **disk_repr** (*COT.disks.DiskRepresentation*) – Disk file representation
- **file_id** (*str*) – Identifier string for the file/disk mapping
- **drive_type** (*str*) – 'harddisk' or 'cdrom'
- **disk** (*xml.etree.ElementTree.Element*) – Existing object to overwrite

Returns *xml.etree.ElementTree.Element* – New or updated disk object

add_disk_device (*drive_type, address, name, description, disk, file_obj, ctrl_item, disk_item=None*)

Create a new disk hardware device or overwrite an existing one.

Parameters

- **drive_type** (*str*) – 'harddisk' or 'cdrom'
- **address** (*str*) – Address on controller, such as "1:0" (optional)
- **name** (*str*) – Device name string (optional)
- **description** (*str*) – Description string (optional)
- **disk** (*xml.etree.ElementTree.Element*) – Disk object to map to this device
- **file_obj** (*xml.etree.ElementTree.Element*) – File object to map to this device
- **ctrl_item** (*OVFItem*) – Controller object to serve as parent
- **disk_item** (*OVFItem*) – Existing disk device to update instead of making a new device.

Returns *xml.etree.ElementTree.Element* – New or updated disk device object.

add_file (*file_path, file_id, file_obj=None, disk=None*)

Add a new file object to the VM or overwrite the provided one.

Parameters

- **file_path** (*str*) – Path to file to add
- **file_id** (*str*) – Identifier string for the file in the VM
- **file_obj** (*xml.etree.ElementTree.Element*) – Existing file object to overwrite
- **disk** (*xml.etree.ElementTree.Element*) – Existing disk object referencing file.

Returns *xml.etree.ElementTree.Element* – New or updated file object

check_sanity_of_disk_device (*disk*, *file_obj*, *disk_item*, *ctrl_item*)

Check if the given disk is linked properly to the other objects.

Parameters

- **disk** (*xml.etree.ElementTree.Element*) – Disk object to validate
- **file_obj** (*xml.etree.ElementTree.Element*) – File object which this disk should be linked to (optional)
- **disk_item** (*OVFItem*) – Disk device object which should link to this disk (optional)
- **ctrl_item** (*OVFItem*) – Controller device object which should link to the disk_item

Raises

- *ValueMismatchError* – if the given items are not linked properly.
- *ValueUnsupportedError* – if the disk_item has a *HostResource* value in an unrecognized or invalid format.

config_file_to_properties (*file_path*, *user_configurable=None*)

Import each line of a text file into a configuration property.

Parameters

- **file_path** (*str*) – File name to import.
- **user_configurable** (*bool*) – Should the resulting properties be configurable at deployment time by the user?

Raises *NotImplementedError* – if the *platform* for this OVF does not define *LITERAL_CLI_STRING*

convert_disk_if_needed (*disk_image*, *kind*)

Convert the disk to a more appropriate format if needed.

- All hard disk files are converted to stream-optimized VMDK as it is the only format that VMware supports in OVA packages.
- CD-ROM iso images are accepted without change.

Parameters

- **disk_image** (*COT.disks.DiskRepresentation*) – Image to inspect and possibly convert
- **kind** (*str*) – Image type (harddisk/cdrom)

Returns *DiskRepresentation* – disk_image, if no conversion was required, or a new *DiskRepresentation* instance representing a converted image that has been created in *output_dir*.

create_configuration_profile (*pid*, *label*, *description*)

Create or update a configuration profile with the given ID.

Parameters

- **pid** (*str*) – Profile identifier
- **label** (*str*) – Brief descriptive label for the profile
- **description** (*str*) – Verbose description of the profile

create_network (*label*, *description*)

Define a new network with the given label and description.

Also serves to update the description of an existing network label.

Parameters

- **label** (*str*) – Brief label for the network
- **description** (*str*) – Verbose description of the network

delete_configuration_profile (*profile*)

Delete the profile with the given ID.

Parameters **profile** (*str*) – Profile ID to delete.

Raises `LookupError` – if the profile does not exist.

static detect_type_from_name (*filename*)

Check the given filename to see if it looks like a type we support.

For our purposes, the file needs to match “.ov[af]” to appear to be an OVF/OVA file. We also support names like “foo.ovf.20150101” as those have been seen in the wild.

Does not check file contents, as the given filename may not yet exist.

Parameters **filename** (*str*) – File name/path

Returns *str* – ‘.ovf’, ‘.box’ or ‘.ova’

Raises `ValueUnsupportedError` – if filename doesn’t match ovf/ova

device_info_str (*device_item*)

Get a one-line summary of a hardware device.

Parameters **device_item** (`OVFItem`) – Device to summarize

Returns *str* – Descriptive string such as “harddisk @ IDE 1:0”

find_device_location (*device*)

Find the controller type and address of a given device object.

Parameters **device** (`OVFItem`) – Hardware device object.

Returns *tuple* – (type, address), such as (“ide”, “1:0”).

Raises `LookupError` – if the controller is not found.

find_disk_from_file_id (*file_id*)

Find the Disk that uses the given file_id for backing.

Parameters **file_id** (*str*) – File identifier string

Returns `xml.etree.ElementTree.Element` – Disk matching the file, or None

find_empty_drive (*drive_type*)

Find a disk device that exists but contains no data.

Parameters **drive_type** (*str*) – Either ‘cdrom’ or ‘harddisk’

Returns `OVFItem` – Instance representing this disk device, or None.

Raises `ValueUnsupportedError` – if drive_type is unrecognized.

find_item_from_disk (*disk*)

Find the disk Item that references the given Disk.

Parameters **disk** (`xml.etree.ElementTree.Element`) – Disk element

Returns *OVFItem* – Corresponding instance, or None

find_item_from_file (*file_obj*)

Find the disk Item that references the given File.

Parameters *file_obj* (*xml.etree.ElementTree.Element*) – File element

Returns *OVFItem* – Corresponding instance, or None.

find_open_controller (*controller_type*)

Find the first open slot on a controller of the given type.

Parameters *controller_type* (*str*) – 'ide' or 'scsi'

Returns *tuple* – (ctrl_item, address_string) or (None, None)

find_parent_from_item (*item*)

Find the parent Item of the given Item.

Parameters *item* (*OVFItem*) – Item whose parent is desired

Returns *OVFItem* – instance representing the parent device, or None

generate_manifest (*ovf_file*)

Construct the manifest file for this package, if possible.

Parameters *ovf_file* (*str*) – OVF descriptor file path

Returns *bool* – True if the manifest was successfully generated, False if not successful (such as if checksum helper tools are unavailable).

get_capacity_from_disk (*disk*)

Get the capacity of the given Disk in bytes.

Parameters *disk* (*xml.etree.ElementTree.Element*) – Disk element to inspect

Returns *int* – Disk capacity, in bytes

get_common_subtype (*device_type*)

Get the subtype common to all devices of the given type.

Parameters *device_type* (*str*) – Device type such as 'ide' or 'memory'.

Returns *str* – Subtype string common to all devices of the type, or None, if multiple such devices exist and they do not all have the same sub-type.

get_file_ref_from_disk (*disk*)

Get the file reference from the given opaque disk object.

Parameters *disk* (*xml.etree.ElementTree.Element*) – 'Disk' element

Returns *str* – 'fileRef' attribute value of this element

get_id_from_disk (*disk*)

Get the identifier string associated with the given Disk object.

Parameters *disk* (*xml.etree.ElementTree.Element*) – Disk object to inspect

Returns *str* – Disk identifier

get_id_from_file (*file_obj*)

Get the file ID from the given opaque file object.

Parameters *file_obj* (*xml.etree.ElementTree.Element*) – 'File' element

Returns *str* – 'id' attribute value of this element

get_nic_count (*profile_list*)

Get the number of NICs under the given profile(s).

Parameters **profile_list** (*list*) – Profile(s) of interest.

Returns *dict* – { profile_name : nic_count }

get_path_from_file (*file_obj*)

Get the file path from the given opaque file object.

Parameters **file_obj** (*xml.etree.ElementTree.Element*) – ‘File’ element

Returns *str* – ‘href’ attribute value of this element

get_property_value (*key*)

Get the value of the given property.

Parameters **key** (*str*) – Property identifier

Returns *str* – Value of this property as a string, or None

get_serial_connectivity (*profile*)

Get the serial port connectivity strings under the given profile.

Parameters **profile** (*str*) – Profile of interest.

Returns *list* – connectivity strings

get_serial_count (*profile_list*)

Get the number of serial ports under the given profile(s).

Parameters **profile_list** (*list*) – Profile(s) of interest.

Returns *dict* – { profile_name : serial_count }

info_string (*width=79, verbosity_option=None*)

Get a descriptive string summarizing the contents of this OVF.

Parameters

- **width** (*int*) – Line length to wrap to where possible.
- **verbosity_option** (*str*) – ‘brief’, None (default), or ‘verbose’

Returns *str* – Wrapped, appropriately verbose string.

profile_info_list (*width=79, verbose=False*)

Get a list describing available configuration profiles.

Parameters

- **width** (*int*) – Line length to wrap to if possible
- **verbose** (*bool*) – if True, generate multiple lines per profile

Returns *tuple* – (header, list)

profile_info_string (*width=79, verbosity_option=None*)

Get a string summarizing available configuration profiles.

Parameters

- **width** (*int*) – Line length to wrap to if possible
- **verbosity_option** (*str*) – ‘brief’, None (default), or ‘verbose’

Returns *str* – Appropriately formatted and verbose string.

remove_file (*file_obj*, *disk=None*, *disk_drive=None*)

Remove the given file object from the VM.

Parameters

- **file_obj** (*xml.etree.ElementTree.Element*) – File object to remove
- **disk** (*xml.etree.ElementTree.Element*) – Disk object referencing file
- **disk_drive** (*OVFItem*) – Disk drive mapping file to a device

Raises *ValueUnsupportedError* – If the *disk_drive* is a device type other than 'cdrom' or 'harddisk'

search_from_controller (*controller*, *address*)

From the controller type and device address, look for existing disk.

This implementation uses the parameters to find matching controller and disk *Item* elements, then using the disk *Item* to find matching *File* and/or *Disk*.

Parameters

- **controller** (*str*) – 'ide' or 'scsi'
- **address** (*str*) – Device address such as '1:0'

Returns *tuple* – (*file*, *disk*, *ctrl_item*, *disk_item*), any or all of which may be *None*

search_from_file_id (*file_id*)

From the given file ID, try to find any existing objects.

This implementation uses the given *file_id* to find a matching *File* in the OVF, then using that to find a matching *Disk* and *Item* entries.

Parameters **file_id** (*str*) – File ID to search from

Returns

tuple – (*file*, *disk*, *ctrl_item*, *disk_item*), any or all of which may be *None*

Raises

- *LookupError* – If the disk entry is found but no corresponding file is found.
- *LookupError* – If the *disk_item* is found but no *ctrl_item* is found to be its parent.

search_from_filename (*filename*)

From the given filename, try to find any existing objects.

This implementation uses the given *filename* to find a matching *File* in the OVF, then using that to find a matching *Disk* and *Item* entries.

Parameters **filename** (*str*) – Filename to search from

Returns *tuple* – (*file*, *disk*, *ctrl_item*, *disk_item*), any or all of which may be *None*

Raises *LookupError* – If the *disk_item* is found but no *ctrl_item* is found to be its parent.

set_capacity_of_disk (*disk*, *capacity_bytes*)

Set the storage capacity of the given *Disk*.

Tries to use the most human-readable form possible (i.e., 8 GiB instead of 8589934592 bytes).

Parameters

- **disk** (*xml.etree.ElementTree.Element*) – Disk to update
- **capacity_bytes** (*int*) – Disk capacity, in bytes

set_cpu_count (*cpus, profile_list*)

Set the number of CPUs.

Parameters

- **cpus** (*int*) – Number of CPUs
- **profile_list** (*list*) – Change only the given profiles

set_ide_subtypes (*type_list, profile_list*)

Set the device subtype(s) for the IDE controller(s).

Parameters

- **type_list** (*list*) – IDE subtype string list
- **profile_list** (*list*) – Change only the given profiles

set_memory (*megabytes, profile_list*)

Set the amount of RAM, in megabytes.

Parameters

- **megabytes** (*int*) – Memory value, in megabytes
- **profile_list** (*list*) – Change only the given profiles

set_nic_count (*count, profile_list*)

Set the given profile(s) to have the given number of NICs.

Parameters

- **count** (*int*) – number of NICs
- **profile_list** (*list*) – Change only the given profiles

set_nic_mac_addresses (*mac_list, profile_list*)

Set the MAC addresses for NICs under the given profile(s).

Note: If the length of `mac_list` is less than the number of NICs, will use the last entry in the list for all remaining NICs.

Parameters

- **mac_list** (*list*) – List of MAC addresses to assign to NICs
- **profile_list** (*list*) – Change only the given profiles

set_nic_names (*name_list, profile_list*)

Set the device names for NICs under the given profile(s).

Parameters

- **name_list** (*list*) – List of names to assign.
- **profile_list** (*list*) – Change only the given profiles

set_nic_networks (*network_list*, *profile_list*)

Set the NIC to network mapping for NICs under the given profile(s).

Note: If the length of *network_list* is less than the number of NICs, will use the last entry in the list for all remaining NICs.

Parameters

- **network_list** (*list*) – List of networks to map NICs to
- **profile_list** (*list*) – Change only the given profiles

set_nic_types (*type_list*, *profile_list*)

Set the hardware type(s) for NICs.

Parameters

- **type_list** (*list*) – NIC hardware type(s)
- **profile_list** (*list*) – Change only the given profiles.

set_property_value (*key*, *value*, *user_configurable=None*, *property_type=None*, *label=None*, *description=None*)

Set the value of the given property (converting value if needed).

Parameters

- **key** (*str*) – Property identifier
- **value** (*object*) – Value to set for this property
- **user_configurable** (*bool*) – Should this property be configurable at deployment time by the user?
- **property_type** (*str*) – Value type - ‘string’ or ‘boolean’
- **label** (*str*) – Brief explanatory label for this property
- **description** (*str*) – Detailed description of this property

Returns *str* – the (converted) value that was set.

Raises `NotImplementedError` – if *ovf_version* is less than 1.0; OVF version 0.9 is not currently supported.

set_scsi_subtypes (*type_list*, *profile_list*)

Set the device subtype(s) for the SCSI controller(s).

Parameters

- **type_list** (*list*) – SCSI subtype string list
- **profile_list** (*list*) – Change only the given profiles

set_serial_connectivity (*conn_list*, *profile_list*)

Set the serial port connectivity under the given profile(s).

Parameters

- **conn_list** (*list*) – List of connectivity strings
- **profile_list** (*list*) – Change only the given profiles

set_serial_count (*count*, *profile_list*)

Set the given profile(s) to have the given number of serial ports.

Parameters

- **count** (*int*) – Number of serial ports
- **profile_list** (*list*) – Change only the given profiles

tar (*ovf_descriptor*, *tar_file*)

Create a .ova tar file based on the given OVF descriptor.

Parameters

- **ovf_descriptor** (*str*) – File path for an OVF descriptor
- **tar_file** (*str*) – File path for the desired OVA archive.

untar (*file_path*)

Untar the OVF descriptor from an .ova to the working directory.

Parameters **file_path** (*str*) – OVA file path

Returns *str* – Path to extracted OVF descriptor

Raises `VMInitError` – if the given file doesn't represent a valid OVA archive.

validate_and_update_file_references ()

Check all File entries to make sure they are valid and up to date.

Helper method for `write()`.

validate_and_update_networks ()

Make sure all defined networks are actually used by NICs.

Delete any networks that are unused and warn the user. Helper method for `write()`.

validate_hardware ()

Check sanity of hardware properties for this VM/product/platform.

Returns *bool* – True if hardware is sane, False if not.

write ()

Write OVF or OVA to *output_file*, if set.

INFO_STRING_DISK_COLUMNS_WIDTH = 41

INFO_STRING_DISK_TEMPLATE = '{{0:{0}}} {{1:>9}} {{2:>9}} {{3:.20}}'

INFO_STRING_FILE_TEMPLATE = '{{0:{0}}} {{1:>9}}'

PROFILE_INFO_TEMPLATE = '{{0:{0}}} {{1:>4}} {{2:>9}} {{3:>4}} {{4:>7}} {{5:>14}}'

application_url

Application URL string (XML AppUrl element).

config_profiles

The list of supported configuration profiles.

If this OVF has no defined profiles, returns an empty list. If there is a default profile, it will be first in the list.

environment_properties

The array of environment properties.

Array of dicts (one per property) with the keys "key", "value", "qualifiers", "type", "user_configurable", "label", and "description".

environment_transports

The list of environment transport method strings.

network_descriptions

The list of network descriptions currently defined in this VM.

Returns *list* – List of network description strings

networks

The list of network names currently defined in this VM.

output_file

OVF or OVA file that will be created or updated by `write()`.

Raises `ValueUnsupportedError` – if `detect_type_from_name()` fails

ovf_version

Float representing the OVF specification version in use.

Supported values at present are 0.9, 1.0, and 2.0.

platform

The platform type, as determined from the OVF descriptor.

This will be the class `GenericPlatform` or a more-specific subclass if recognized as such.

product

Short descriptive product string (XML `Product` element).

product_class

The product class identifier, such as `com.cisco.csr1000v`.

product_url

Product URL string (XML `ProductUrl` element).

system_types

List of virtual system type(s) supported by this virtual machine.

For an OVF, this corresponds to the `VirtualSystemType` element.

vendor

Short descriptive vendor string (XML `Vendor` element).

vendor_url

Vendor URL string (XML `VendorUrl` element).

version_long

Long descriptive version string (XML `FullVersion` element).

version_short

Short descriptive version string (XML `Version` element).

byte_count (*base_val*, *multiplier*)

Convert an OVF-style value + multiplier into decimal byte count.

Inverse operation of `factor_bytes()`.

Parameters

- **base_val** (*str*) – Base value string (value of `ovf:capacity`, etc.)
- **multiplier** (*str*) – Multiplier string (value of `ovf:capacityAllocationUnits`, etc.)

Returns *int* – Number of bytes

Examples

```
>>> byte_count("128", "byte * 2^20")
134217728
>>> byte_count("512", "MegaBytes")
536870912
```

byte_string (*byte_value*, *base_shift*=0)

Pretty-print the given bytes value.

Parameters

- **byte_value** (*float*) – Value
- **base_shift** (*int*) – Base value of byte_value (0 = bytes, 1 = KiB, 2 = MiB, etc.)

Returns *str* – Pretty-printed byte string such as “1.00 GiB”

Examples

```
>>> byte_string(512)
'512 B'
>>> byte_string(512, 2)
'512 MiB'
>>> byte_string(65536, 2)
'64 GiB'
>>> byte_string(65547)
'64.01 KiB'
>>> byte_string(65530, 3)
'63.99 TiB'
>>> byte_string(1023850)
'999.9 KiB'
>>> byte_string(1024000)
'1000 KiB'
>>> byte_string(1048575)
'1024 KiB'
>>> byte_string(1049200)
'1.001 MiB'
>>> byte_string(2560)
'2.5 KiB'
```

factor_bytes (*byte_value*)

Convert a byte count into OVF-style bytes + multiplier.

Inverse operation of *byte_count* ()

Parameters **byte_value** (*int*) – Number of bytes

Returns *tuple* – (base_val, multiplier)

Examples

```
>>> factor_bytes(134217728)
('128', 'byte * 2^20')
>>> factor_bytes(134217729)
('134217729', 'byte')
```

COT.ovf.hardware module

Representation of OVF hardware definitions.

Classes and Exceptions

<i>OVFHardware</i>	Helper class for <i>OVF</i> .
<i>OVFHardwareDataError</i>	The input data used to construct an <i>OVFHardware</i> is not sane.

exception OVFHardwareDataError

Bases: `exceptions.Exception`

The input data used to construct an *OVFHardware* is not sane.

class OVFHardware (ovf)

Bases: `object`

Helper class for *OVF*.

Represents all hardware items defined by this OVF; i.e., the contents of all Items in the VirtualHardwareSection.

Fundamentally it's just a dict of *OVFItem* objects with a bunch of helper methods.

__init__ (ovf)

Construct an OVFHardware object describing all Items in the OVF.

Parameters *ovf* (*OVF*) – OVF instance to extract hardware information from.

Raises *OVFHardwareDataError* – if any data errors are seen

clone_item (parent_item, profile_list)

Clone an OVFItem to create a new instance.

Parameters

- **parent_item** (*OVFItem*) – Instance to clone from
- **profile_list** (*list*) – List of profiles to clone into

Returns *tuple* – (instance_id, ovfitem)

delete_item (item)

Delete the given Item from the hardware.

Parameters *item* (*OVFItem*) – Item to delete

find_all_items (resource_type=None, properties=None, profile_list=None)

Find all items matching the given type, properties, and profiles.

Parameters

- **resource_type** (*str*) – Resource type string like 'scsi' or 'serial'
- **properties** (*dict*) – Properties and their values to match
- **profile_list** (*list*) – List of profiles to filter on

Returns *list* – Matching *OVFItem* instances

find_item (resource_type=None, properties=None, profile=None)

Find the only OVFItem of the given resource_type.

Parameters

- **resource_type** (*str*) – Resource type string like 'scsi' or 'serial'

- **properties** (*dict*) – Properties and their values to match
- **profile** (*str*) – Single profile ID to search within

Returns *OVFItem* – Matching instance, or None

Raises *LookupError* – if more than one such Item exists.

find_unused_instance_id()

Find the first available InstanceID number.

Returns *str* – An instance ID that is not yet in use.

get_item_count (*resource_type*, *profile*)

Wrapper for *get_item_count_per_profile()*.

Parameters

- **resource_type** (*str*) – Resource type string like ‘scsi’ or ‘serial’
- **profile** (*str*) – Single profile identifier string to look up.

Returns *int* – Number of items of this type in this profile.

get_item_count_per_profile (*resource_type*, *profile_list*)

Get the number of Items of the given type per profile.

Items present under “no profile” will be counted against the total for each profile.

Parameters

- **resource_type** (*str*) – Resource type string like ‘scsi’ or ‘serial’
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)

Returns *dict* – mapping profile strings to the number of items under each profile.

item_match (*item*, *resource_type*, *properties*, *profile_list*)

Check whether the given item matches the given filters.

Parameters

- **item** (*OVFItem*) – Item to validate
- **resource_type** (*str*) – Resource type string like ‘scsi’ or ‘serial’
- **properties** (*dict*) – Properties and their values to match
- **profile_list** (*list*) – List of profiles to filter on

Returns *bool* – True if the item matches all filters, False if not.

new_item (*resource_type*, *profile_list=None*)

Create a new *OVFItem* of the given type.

Parameters

- **resource_type** (*str*) – String such as ‘cpu’ or ‘harddisk’ - used as a key to *RES_MAP*
- **profile_list** (*list*) – Profiles the new item should belong to

Returns *tuple* – (instance_id, ovfitem)

set_item_count_per_profile (*resource_type*, *count*, *profile_list*)

Set the number of items of a given type under the given profile(s).

If the new count is greater than the current count under this profile, then additional instances that already exist under another profile will be added to this profile, starting with the lowest-sequence instance not already present, and only as a last resort will new instances be created.

If the new count is less than the current count under this profile, then the highest-numbered instances will be removed preferentially.

Parameters

- **resource_type** (*str*) – ‘cpu’, ‘harddisk’, etc.
- **count** (*int*) – Desired number of items
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)

set_item_values_per_profile (*resource_type*, *prop_name*, *value_list*, *profile_list*, *default=None*)

Set value(s) for a property of multiple items of a type.

Parameters

- **resource_type** (*str*) – Device type such as ‘harddisk’ or ‘cpu’
- **prop_name** (*str*) – Property name to update
- **value_list** (*list*) – List of values to set (one value per item of the given resource_type)
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)
- **default** (*str*) – If there are more matching items than entries in value_list, set extra items to this value

set_value_for_all_items (*resource_type*, *prop_name*, *new_value*, *profile_list*, *create_new=False*)

Set a property to the given value for all items of the given type.

If no items of the given type exist, will create a new Item if create_new is set to True; otherwise will log a warning and do nothing.

Parameters

- **resource_type** (*str*) – Resource type such as ‘cpu’ or ‘harddisk’
- **prop_name** (*str*) – Property name to update
- **new_value** (*str*) – New value to set the property to
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)
- **create_new** (*bool*) – Whether to create a new entry if no items of this resource_type presently exist.

update_existing_item_count_per_profile (*resource_type*, *count*, *profile_list*)

Change profile membership of existing items as needed.

Helper method for `set_item_count_per_profile()`.

Parameters

- **resource_type** (*str*) – ‘cpu’, ‘harddisk’, etc.
- **count** (*int*) – Desired number of items
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)

Returns *tuple* – (count_dict, items_to_add, last_item)

update_xml ()

Regenerate all Items under the VirtualHardwareSection, if needed.

Will do nothing if no Items have been changed.

COT.ovf.item module

Module for working with individual hardware elements in an OVF.

Represents all variations of a given hardware `Item` amongst different hardware configuration profiles.

Functions

<code>list_union</code>	Get union of lists.
-------------------------	---------------------

Classes and Exceptions

<code>OVFItem</code>	Helper class for OVF.
<code>OVFItemDataError</code>	Data to be added to an <code>OVFItem</code> conflicts with existing data.

exception OVFItemDataError

Bases: `exceptions.Exception`

Data to be added to an `OVFItem` conflicts with existing data.

class OVFItem (ovf, item=None)

Bases: `object`

Helper class for OVF.

Represents all variations of a given hardware `Item` amongst different hardware configuration profiles.

In essence, it is:

- a dict of `Item` properties (indexed by element name)
- each of which is a dict of sets of profiles (indexed by element value)

__init__ (ovf, item=None)

Create a new `OVFItem` with contents based on the given `Item` element.

Parameters

- **ovf** (`OVF`) – OVF instance that owns the `Item` (optional)
- **item** (`xml.etree.ElementTree.Element`) – ‘Item’ element (optional)

add_item (item)

Add the given `Item` element to this `OVFItem`.

Parameters **item** (`xml.etree.ElementTree.Element`) – XML `Item` element

Raises

- `ValueUnsupportedError` – if the `item` is not a recognized `Item` variant.
- `OVFItemDataError` – if the new `Item` conflicts with existing data already in the `OVFItem`.

add_profile (new_profile, from_item=None)

Add a new profile to this item.

Parameters

- **new_profile** (`str`) – Profile name to add
- **from_item** (`OVFItem`) – Item to inherit properties from. If unset, this defaults to `self`.

Raises `RuntimeError` – If unable to determine what value to inherit for a particular property.

all_profiles (*name*, *default=None*)

Superset of all profiles for which this name has a value.

Parameters

- **name** (*str*) – Property name.
- **default** (*object*) – Default value to return if there are no matches

Returns Set of profile strings, or the given *default* if no matches.

generate_items ()

Get a list of Item XML elements derived from this object's data.

Returns *list* – Generated list of XML Item elements

get (*tag*)

Get the dict associated with the given XML tag, if any.

Parameters **tag** (*str*) – XML tag to look up

Returns *dict* – Dictionary of values associated with this tag (TODO?)

get_all_values (*tag*)

Get the list of all value strings for the given tag.

Parameters **tag** (*str*) – Tag to retrieve value for

Returns *list* – List of value strings.

get_nonintersecting_set_list ()

Identify the minimal non-intersecting set of profiles.

Returns *list* – List of profile-set strings.

get_value (*tag*, *profiles=None*)

Get the value for the given tag under the given profiles.

If the tag does not exist under these profiles, or the tag values differ across the profiles, returns *None*.

Parameters

- **tag** (*str*) – Tag to retrieve value for
- **profiles** (*set*) – set of profile names, or *None*

Returns Value string or list, or *None*

Raises *OVFItemDataError* – if *value_replace_wildcards()* failed to remove any wildcards from the internally stored value.

has_profile (*profile*)

Check if this Item exists under the given profile.

Parameters **profile** (*str*) – Profile name

Returns *bool* – True if the item exists in this profile, False if not.

property_profiles (*name*, *value*)

Get set of profiles associated with a property name and value.

Parameters

- **name** (*str*) – Property name.
- **value** (*object*) – Property value of interest.

Returns *set* – Profile strings associated with this name/value.

property_values (*name*)

Get list of values known for a given property name.

Parameters *name* (*str*) – Property name.

Returns *list* – List of values

remove_profile (*profile*, *split_default=True*)

Remove all trace of the given profile from this item.

Parameters

- **profile** (*str*) – Profile name to remove
- **split_default** (*bool*) – If False, do not split out ‘default’ profile items to specifically exclude this profile. Used when the profile being removed will no longer exist anywhere and so ‘default’ will continue to exclude this profile.

set_property (*name*, *value*, *profiles=None*, *overwrite=True*)

Store the value and profiles associated with it for the given name.

Parameters

- **name** (*str*) – Property name
- **value** (*str*) – Value associated with *name*
- **profiles** (*list*) – If *None*, set for all profiles currently known to this item, else set only for the given list of profiles.
- **overwrite** (*bool*) – Whether to permit overwriting of existing value set in this item.

Raises *OVFItemDataError* – if a value is already defined and would be overwritten, unless *overwrite* is *True*

validate ()

Verify that the OVFItem describes a valid set of items.

Also clean up any oddities (like a property value assigned to ‘all profiles’ and also redundantly to a specific profile).

Raises *RuntimeError* – if validation fails and COT doesn’t know how to automatically repair the error(s) identified.

value_add_wildcards (*name*, *value*, *profiles*)

Add wildcard placeholders to a string that may need updating.

If the Description references the ElementName, or the ElementName or Description references the VirtualQuantity, Connection, or ResourceSubType, replace such references with a placeholder that we can regenerate at output time. That way, if any of the linked items change, these strings can change too.

Parameters

- **name** (*str*) – Property name
- **value** (*str*) – Value to add wildcards to.
- **profiles** (*list*) – Profiles to which this (name, value) applies.

Returns *str* – The updated value string with wildcards added.

See also:

value_replace_wildcards ()

value_replace_wildcards (*name*, *value*, *profiles*)

Replace wildcards with actual values.

Parameters

- **name** (*str*) – Property name
- **value** (*str*) – Value to replace wildcards from.
- **profiles** (*list*) – Profiles to which this (name, value) applies.

Returns *str* – The updated value string, with wildcards replaced.

See also:

`value_add_wildcards()`

ATTRIB_KEY_SUFFIX = ‘{Item attribute}’

ELEMENT_KEY_SUFFIX = ‘{custom element}’

hardware_subtype

Device hardware subtype such as ‘virtio’ or ‘lsilogic’.

hardware_type

Device hardware type such as ‘ide’ or ‘memory’.

instance_id

Device instance ID.

properties = None

Dict of dicts. properties[name][value] = (profile1, profile2).

property_names

List of names of all properties known to this OVFIItem.

list_union (**lists*)

Get union of lists.

Parameters **lists** (*list*) – List of lists to unify.

Returns *list* – All distinct values across the given lists.

Examples

```
>>> list_union([1, 2, 3], [0, 4], [1, 5])
[1, 2, 3, 0, 4, 5]
>>> list_union(['foo'], ['bar'], ['bar', 'foo'])
['foo', 'bar']
>>> list_union(['bar', 'foo'], ['foo'], ['bar'])
['bar', 'foo']
```

COT.ovf.name_helper module

Module for handling the differences in XML between OVF spec versions.

Functions

<code>name_helper</code>	Generate an instance of the correct OVFFNameHelper variant class.
--------------------------	---

Classes and Exceptions

<code>OVFNameHelper1</code>	Helper class for OVF version 1.x.
<code>OVFNameHelper0</code>	Helper class for OVF of versions prior to 1.0.
<code>OVFNameHelper2</code>	Helper class for OVF of version 2.x.

class OVFNameHelper0

Bases: `COT.ovf.name_helper.OVFNameHelper1`

Helper class for OVF of versions prior to 1.0.

Provides string constants for easier lookup of various OVF XML elements and attributes.

`__init__()`

Create a name helper for OVF version 0.x.

NSM = {'vssd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData', 'rasd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData'}

Shorthand for XML namespace URIs usually seen in a version 0.x OVF.

class OVFNameHelper1

Bases: `object`

Helper class for OVF version 1.x.

Provides string constants for easier lookup of various OVF XML elements and attributes.

Version-specific subclasses below provide variant properties.

`__init__()`

Create a name helper for OVF version 1.x.

`item_tag_for_namespace(ns)`

Get the Item tag for the given XML namespace.

Parameters `ns` (*str*) – XML namespace

Returns *str* – 'Item', 'StorageItem', or 'EthernetPortItem' as appropriate.

Raises `ValueUnsupportedError` – if the namespace is unrecognized

`namespace_for_item_tag(tag)`

Get the XML namespace for the given item tag.

Parameters `tag` (*str*) – Un-namespaced XML tag.

Returns *str* – XML namespace string, or None.

`namespace_for_resource_type(resource_type)`

Get the XML namespace for the given ResourceType.

Parameters `resource_type` (*str*) – ResourceType value string.

Returns *str* – XML namespace string, or None.

NSM = {'vssd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData', 'rasd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData'}

Shorthand for XML namespace URIs usually seen in a version 1.x OVF.

RES_MAP = {'ib': '9', 'usb': '23', 'floppy': '14', 'dvd': '16', 'iscsi': '8', 'harddisk': '17', 'parallel': '22', 'sata': '20', 'fc': '21'}

Mapping of human-readable strings to ResourceType values.

See http://schemas.dmtf.org/wbem/cim-html/2/CIM_ResourceAllocationSettingData.html for more details.

class OVFNameHelper2

Bases: `COT.ovf.name_helper.OVFNameHelper1`

Helper class for OVF of version 2.x. TODO.

Provides string constants for easier lookup of various OVF XML elements and attributes.

`__init__()`

Create a name helper for OVF version 2.x.

NSM = {'vssd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData', 'rasd': 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData'}

Shorthand for XML namespace URIs usually seen in a version 2.x OVF.

name_helper (*version*)

Generate an instance of the correct OV FNameHelper variant class.

Parameters **version** (*float*) – OVF specification version to use, such as 0.9, 1.0, or 2.0

Returns Instance of OV FNameHelper[012] as appropriate.

COT.disks package reference

Package for handling various disk file types (VMDK, ISO, QCOW2, etc.).

Tries to provide an API that abstracts away differences in how the various types need to be operated on (e.g., qemu-img, mkisofs, etc.).

API

<i>convert_disk</i>	Convert a disk representation into a new format.
<i>create_disk</i>	Create a disk of the requested format.
<i>disk_representation_from_file</i>	Get a DiskRepresentation appropriate to the given file.
<i>DiskRepresentation</i>	Abstract disk image file representation.

Disk modules

<i>COT.disks.disk</i>	Abstract base class for representations of disk image files.
<i>COT.disks.iso</i>	Handling of ISO files.
<i>COT.disks.qcow2</i>	Handling of QCOW2 files.
<i>COT.disks.raw</i>	Handling of raw disk image files.
<i>COT.disks.vmdk</i>	Handling of VMDK files.

COT.disks.disk module

Abstract base class for representations of disk image files.

class DiskRepresentation (*path, disk_subformat=None, capacity=None, files=None*)

Bases: `object`

Abstract disk image file representation.

__init__ (*path, disk_subformat=None, capacity=None, files=None*)

Create a representation of an existing disk or create a new disk.

Parameters

- **path** (*str*) – Path to existing file or path to create new file at.

- **disk_subformat** (*str*) – Subformat option(s) of the disk to create (e.g., ‘rockridge’ for ISO, ‘streamOptimized’ for VMDK), if any.
- **capacity** (*int*) – Capacity of disk to create
- **files** (*int*) – Files to place in the filesystem of this disk.

create_file ()

Given parameters but not an existing file, create that file.

classmethod file_is_this_type (*path*)

Check if the given file is image type represented by this class.

Parameters **path** (*str*) – Path to file to check.

Returns *bool* – True (file matches this type) or False (file does not match)

Raises `HelperError` – if no file exists at path.

classmethod from_other_image (*input_image*, *output_dir*, *output_subformat=None*)

Convert the other disk image into an image of this type.

Parameters

- **input_image** (`DiskRepresentation`) – Existing image representation.
- **output_dir** (*str*) – Output directory to store the new image in.
- **output_subformat** (*str*) – Any relevant subformat information.

Raises `NotImplementedError` – Subclasses may implement this.

capacity

Capacity of this disk image, in bytes.

disk_format = None

Disk format represented by this class.

disk_subformat

Sub-format of the disk, such as ‘rockridge’ or ‘streamOptimized’.

files

List of files embedded in this disk image.

path

System path to this disk file.

COT.disks.iso module

Handling of ISO files.

class ISO (*path*, *disk_subformat=None*, *capacity=None*, *files=None*)

Bases: `COT.disks.disk.DiskRepresentation`

ISO 9660 disk image file representation.

classmethod file_is_this_type (*path*)

Detect whether the given file is an ISO image.

Parameters **path** (*str*) – Path to file

Returns *bool* – True (file is an ISO) or False (file is not an ISO)

Raises `HelperError` – if path is not a file at all.

classmethod from_other_image (*input_image*, *output_dir*, *output_subformat=None*)

Convert the other disk image into an image of this type.

Parameters

- **input_image** (*DiskRepresentation*) – Existing image representation.
- **output_dir** (*str*) – Output directory to store the new image in.
- **output_subformat** (*str*) – Any relevant subformat information.

Raises *NotImplementedError* – non-trivial to convert other types to ISO

disk_format = 'iso'

disk_subformat

ISO sub-format.

Possible values:

- "" - not Rock Ridge
- "rockridge" - has Rock Ridge extensions

files

The list of files contained in this ISO.

COT.disks.qcow2 module

Handling of QCOW2 files.

class QCOW2 (*path*, *disk_subformat=None*, *capacity=None*, *files=None*)

Bases: *COT.disks.disk.DiskRepresentation*

QCOW2 disk image file representation.

classmethod from_other_image (*input_image*, *output_dir*, *output_subformat=None*)

Convert the other disk image into an image of this type.

Parameters

- **input_image** (*DiskRepresentation*) – Existing image representation.
- **output_dir** (*str*) – Output directory to store the new image in.
- **output_subformat** (*str*) – Any relevant subformat information.

Returns *QCOW2* – representation of newly created qcow2 image file

disk_format = 'qcow2'

COT.disks.raw module

Handling of raw disk image files.

class RAW (*path*, *disk_subformat=None*, *capacity=None*, *files=None*)

Bases: *COT.disks.disk.DiskRepresentation*

Raw disk image file representation.

classmethod from_other_image (*input_image*, *output_dir*, *output_subformat=None*)

Convert the other disk image into an image of this type.

Parameters

- **input_image** (*DiskRepresentation*) – Existing image representation.
- **output_dir** (*str*) – Output directory to store the new image in.
- **output_subformat** (*str*) – Any relevant subformat information.

Returns *RAW* – representation of newly created raw image.

disk_format = 'raw'

files

List of files on the FAT32 file system of this disk.

COT.disks.vmdk module

Handling of VMDK files.

class VMDK (*path, disk_subformat=None, capacity=None, files=None*)

Bases: *COT.disks.disk.DiskRepresentation*

VMDK disk image file representation.

classmethod from_other_image (*input_image, output_dir, output_subformat='streamOptimized'*)

Convert the other disk image into an image of this type.

Parameters

- **input_image** (*DiskRepresentation*) – Existing image representation.
- **output_dir** (*str*) – Output directory to store the new image in.
- **output_subformat** (*str*) – VMDK subformat string. Defaults to “streamOptimized” if unset.

Returns *VMDK* – representation of newly created VMDK file.

disk_format = 'vmdk'

disk_subformat

Disk subformat, such as 'streamOptimized'.

convert_disk (*disk_image, new_directory, new_format, new_subformat=None*)

Convert a disk representation into a new format.

Parameters

- **disk_image** (*DiskRepresentation*) – Existing disk image as input.
- **new_directory** (*str*) – Directory to create new image under
- **new_format** (*str*) – Format to convert to.
- **new_subformat** (*str*) – (optional) Sub-format to convert to.

Returns *DiskRepresentation* – Converted disk.

create_disk (*disk_format, *args, **kwargs*)

Create a disk of the requested format.

Parameters **disk_format** (*str*) – Disk format such as 'iso' or 'vmdk'.

For the other parameters, see *DiskRepresentation*.

Returns *DiskRepresentation* – Created disk

disk_representation_from_file (*file_path*)

Get a *DiskRepresentation* appropriate to the given file.

Parameters `file_path` (*str*) – Path of existing file to represent.

Returns *DiskRepresentation* – Representation of this file.

Indices and tables

- `genindex`
- `modindex`
- `search`

- .
- COT.add_disk, 53
- COT.add_file, 57
- COT.cli, 93
- COT.data_validation, 71
- COT.deploy, 58
- COT.deploy_esxi, 60
- COT.disks, 129
- COT.disks.disk, 129
- COT.disks.iso, 130
- COT.disks.qcow2, 131
- COT.disks.raw, 131
- COT.disks.vmdk, 132
- COT.edit_hardware, 62
- COT.edit_product, 65
- COT.edit_properties, 66
- COT.file_reference, 79
- COT.help, 68
- COT.helpers, 97
- COT.helpers.apr_get, 103
- COT.helpers.brew, 103
- COT.helpers.fatdisk, 104
- COT.helpers.gcc, 104
- COT.helpers.helper, 98
- COT.helpers.isoinfo, 104
- COT.helpers.make, 104
- COT.helpers.mkisofs, 105
- COT.helpers.ovftool, 105
- COT.helpers.port, 106
- COT.helpers.qemu_img, 106
- COT.helpers.vmdktool, 106
- COT.helpers.yum, 107
- COT.info, 68
- COT.inject_config, 69
- COT.install_helpers, 70
- COT.ovf, 107
- COT.ovf.hardware, 120
- COT.ovf.item, 123
- COT.ovf.name_helper, 126
- COT.ovf.ovf, 107
- COT.platforms, 81
- COT.platforms.cisco_csr1000v, 83
- COT.platforms.cisco_iosv, 84
- COT.platforms.cisco_iosxrv, 86
- COT.platforms.cisco_iosxrv_9000, 88
- COT.platforms.cisco_nexus_9000v, 88
- COT.platforms.cisco_nxosv, 90
- COT.platforms.generic, 81
- COT.remove_file, 70
- COT.submodule, 51
- COT.ui_shared, 91
- COT.vm_context_manager, 48
- COT.vm_description, 39
- COT.vm_factory, 48
- COT.xml_file, 48

C

COT, 39

Symbols

[__init__\(\) \(AptGet method\), 103](#)
[__init__\(\) \(Brew method\), 103](#)
[__init__\(\) \(CLI method\), 93](#)
[__init__\(\) \(COTAddDisk method\), 53](#)
[__init__\(\) \(COTAddFile method\), 57](#)
[__init__\(\) \(COTDeploy method\), 58](#)
[__init__\(\) \(COTDeployESXi method\), 61](#)
[__init__\(\) \(COTEditHardware method\), 63](#)
[__init__\(\) \(COTEditProduct method\), 65](#)
[__init__\(\) \(COTEditProperties method\), 66](#)
[__init__\(\) \(COTGenericSubmodule method\), 51](#)
[__init__\(\) \(COTHelp method\), 68](#)
[__init__\(\) \(COTInfo method\), 68](#)
[__init__\(\) \(COTInjectConfig method\), 69](#)
[__init__\(\) \(COTInstallHelpers method\), 70](#)
[__init__\(\) \(COTReadOnlySubmodule method\), 52](#)
[__init__\(\) \(COTRemoveFile method\), 71](#)
[__init__\(\) \(COTSubmodule method\), 52](#)
[__init__\(\) \(DiskRepresentation method\), 129](#)
[__init__\(\) \(FatDisk method\), 104](#)
[__init__\(\) \(FileInTAR method\), 79](#)
[__init__\(\) \(FileOnDisk method\), 80](#)
[__init__\(\) \(GCC method\), 104](#)
[__init__\(\) \(GenISOImage method\), 105](#)
[__init__\(\) \(Helper method\), 99](#)
[__init__\(\) \(HelperDict method\), 101](#)
[__init__\(\) \(ISOInfo method\), 104](#)
[__init__\(\) \(Make method\), 105](#)
[__init__\(\) \(MkISOFS method\), 105](#)
[__init__\(\) \(OVF method\), 108](#)
[__init__\(\) \(OVFHardware method\), 120](#)
[__init__\(\) \(OVFItem method\), 123](#)
[__init__\(\) \(OVFNameHelper0 method\), 127](#)
[__init__\(\) \(OVFNameHelper1 method\), 127](#)
[__init__\(\) \(OVFNameHelper2 method\), 128](#)
[__init__\(\) \(OVFTool method\), 105](#)
[__init__\(\) \(Port method\), 106](#)
[__init__\(\) \(PyVmomiVMReconfigSpec method\), 62](#)
[__init__\(\) \(QEMUImg method\), 106](#)

[__init__\(\) \(SerialConnection method\), 59](#)
[__init__\(\) \(SmarterConnection method\), 62](#)
[__init__\(\) \(UI method\), 91](#)
[__init__\(\) \(VMContextManager method\), 48](#)
[__init__\(\) \(VMDKTool method\), 106](#)
[__init__\(\) \(VMDescription method\), 40](#)
[__init__\(\) \(ValueUnsupportedError method\), 73](#)
[__init__\(\) \(XML method\), 48](#)
[__init__\(\) \(XorrISO method\), 105](#)
[__init__\(\) \(Yum method\), 107](#)
[_install\(\) \(Helper method\), 99](#)
[_provider_package \(Helper attribute\), 100](#)

A

[add_child\(\) \(COT.xml_file.XML class method\), 49](#)
[add_controller_device\(\) \(OVF method\), 108](#)
[add_controller_device\(\) \(VMDescription method\), 40](#)
[add_disk\(\) \(OVF method\), 109](#)
[add_disk\(\) \(VMDescription method\), 40](#)
[add_disk_device\(\) \(OVF method\), 109](#)
[add_disk_device\(\) \(VMDescription method\), 40](#)
[add_disk_worker\(\) \(in module COT.add_disk\), 54](#)
[add_file\(\) \(OVF method\), 109](#)
[add_file\(\) \(VMDescription method\), 41](#)
[add_item\(\) \(OVFItem method\), 123](#)
[add_profile\(\) \(OVFItem method\), 123](#)
[add_subparser\(\) \(CLI method\), 94](#)
[add_to_archive\(\) \(FileInTAR method\), 80](#)
[add_to_archive\(\) \(FileOnDisk method\), 80](#)
[address \(COTAddDisk attribute\), 53](#)
[all_profiles\(\) \(OVFItem method\), 123](#)
[alphanum_split\(\) \(in module COT.data_validation\), 73](#)
[application_url \(COTEditProduct attribute\), 66](#)
[application_url \(OVF attribute\), 117](#)
[AptGet \(class in COT.helpers.apr_get\), 103](#)
[args_to_dict\(\) \(CLI static method\), 94](#)
[ATTRIB_KEY_SUFFIX \(OVFItem attribute\), 126](#)

B

[BOOTSTRAP_DISK_TYPE \(GenericPlatform attribute\),](#)

BOOTSTRAP_DISK_TYPE (IOSv attribute), 85

Brew (class in COT.helpers.brew), 103

byte_count() (in module COT.ovf.ovf), 118

byte_string() (in module COT.ovf.ovf), 119

C

call() (Helper method), 99

canonicalize_helper() (in module COT.data_validation), 73

canonicalize_ide_subtype() (in module COT.data_validation), 73

canonicalize_nic_subtype() (in module COT.data_validation), 74

canonicalize_scsi_subtype() (in module COT.data_validation), 74

capacity (DiskRepresentation attribute), 130

check_call() (in module COT.helpers.helper), 101

check_for_conflict() (in module COT.data_validation), 75

check_output() (in module COT.helpers.helper), 102

check_sanity_of_disk_device() (OVF method), 109

check_sanity_of_disk_device() (VMDescription method), 41

choose_from_list() (UI method), 91

CLI (class in COT.cli), 93

clone_item() (OVFHardware method), 120

close() (FileInTAR method), 80

close() (FileOnDisk method), 80

config_file (COTEditProperties attribute), 67

config_file (COTInjectConfig attribute), 69

config_file_to_properties() (OVF method), 110

config_file_to_properties() (VMDescription method), 41

config_profiles (OVF attribute), 117

config_profiles (VMDescription attribute), 47

CONFIG_TEXT_FILE (CSR1000V attribute), 84

CONFIG_TEXT_FILE (GenericPlatform attribute), 83

CONFIG_TEXT_FILE (IOSv attribute), 85

CONFIG_TEXT_FILE (IOSXRv attribute), 87

CONFIG_TEXT_FILE (IOSXRvLC attribute), 87

CONFIG_TEXT_FILE (Nexus9000v attribute), 89

CONFIG_TEXT_FILE (NXOSv attribute), 90

configuration (COTDeploy attribute), 58

confirm() (CLI method), 94

confirm() (UI method), 92

confirm_elements() (in module COT.add_disk), 54

confirm_or_die() (UI method), 92

controller, 21

controller (COTAddDisk attribute), 54

controller_type_for_device()
(COT.platforms.cisco_csr1000v.CSR1000V
class method), 83

controller_type_for_device()
(COT.platforms.generic.GenericPlatform
class method), 81

convert_disk() (in module COT.disks), 132

convert_disk_if_needed() (OVF method), 110

convert_disk_if_needed() (VMDescription method), 41

copy_to() (FileInTAR method), 80

copy_to() (FileOnDisk method), 80

COT, 21

COT (module), 39

COT.add_disk (module), 53

COT.add_file (module), 57

COT.cli (module), 93

COT.data_validation (module), 71

COT.deploy (module), 58

COT.deploy_esxi (module), 60

COT.disks (module), 129

COT.disks.disk (module), 129

COT.disks.iso (module), 130

COT.disks.qcow2 (module), 131

COT.disks.raw (module), 131

COT.disks.vmdk (module), 132

COT.edit_hardware (module), 62

COT.edit_product (module), 65

COT.edit_properties (module), 66

COT.file_reference (module), 79

COT.help (module), 68

COT.helpers (module), 97

COT.helpers.appt_get (module), 103

COT.helpers.brew (module), 103

COT.helpers.fatdisk (module), 104

COT.helpers.gcc (module), 104

COT.helpers.helper (module), 98

COT.helpers.isoinfo (module), 104

COT.helpers.make (module), 104

COT.helpers.mkisofs (module), 105

COT.helpers.ovftool (module), 105

COT.helpers.port (module), 106

COT.helpers.qemu_img (module), 106

COT.helpers.vmdktool (module), 106

COT.helpers.yum (module), 107

COT.info (module), 68

COT.inject_config (module), 69

COT.install_helpers (module), 70

COT.ovf (module), 107

COT.ovf.hardware (module), 120

COT.ovf.item (module), 123

COT.ovf.name_helper (module), 126

COT.ovf.ovf (module), 107

COT.platforms (module), 81

COT.platforms.cisco_csr1000v (module), 83

COT.platforms.cisco_iosv (module), 84

COT.platforms.cisco_iosxrv (module), 86

COT.platforms.cisco_iosxrv_9000 (module), 88

COT.platforms.cisco_nexus_9000v (module), 88

COT.platforms.cisco_nxosv (module), 90

COT.platforms.generic (module), 81

COT.remove_file (module), 70

COT.submodule (module), 51
 COT.ui_shared (module), 91
 COT.vm_context_manager (module), 48
 COT.vm_description (module), 39
 COT.vm_factory (module), 48
 COT.xml_file (module), 48
 COTAddDisk (class in COT.add_disk), 53
 COTAddFile (class in COT.add_file), 57
 COTDeploy (class in COT.deploy), 58
 COTDeployESXi (class in COT.deploy_esxi), 61
 COTEditHardware (class in COT.edit_hardware), 63
 COTEditProduct (class in COT.edit_product), 65
 COTEditProperties (class in COT.edit_properties), 66
 COTGenericSubmodule (class in COT.submodule), 51
 COTHelp (class in COT.help), 68
 COTInfo (class in COT.info), 68
 COTInjectConfig (class in COT.inject_config), 69
 COTInstallHelpers (class in COT.install_helpers), 70
 COTReadOnlySubmodule (class in COT.submodule), 52
 COTRemoveFile (class in COT.remove_file), 70
 COTSubmodule (class in COT.submodule), 52
 cp() (Helper static method), 99
 cpus (COTEditHardware attribute), 63
 create() (COT.vm_factory.VMFactory class method), 48
 create_configuration_profile() (OVF method), 110
 create_configuration_profile() (VMDescription method), 41
 create_disk() (in module COT.disks), 132
 create_file() (DiskRepresentation method), 130
 create_network() (OVF method), 110
 create_network() (VMDescription method), 42
 create_parser() (CLI method), 94
 create_subparser() (COTAddDisk method), 53
 create_subparser() (COTAddFile method), 57
 create_subparser() (COTDeploy method), 58
 create_subparser() (COTDeployESXi method), 61
 create_subparser() (COTEditHardware method), 63
 create_subparser() (COTEditProduct method), 65
 create_subparser() (COTEditProperties method), 66
 create_subparser() (COTGenericSubmodule method), 51
 create_subparser() (COTHelp method), 68
 create_subparser() (COTInfo method), 68
 create_subparser() (COTInjectConfig method), 69
 create_subparser() (COTInstallHelpers method), 70
 create_subparser() (COTRemoveFile method), 71
 create_subparsers() (CLI method), 94
 CSR1000V (class in COT.platforms.cisco_csr1000v), 83

D

datastore (COTDeployESXi attribute), 61
 default_config_profile (VMDescription attribute), 47
 default_confirm_response (UI attribute), 93
 delete_all_other_profiles (COTEditHardware attribute), 63

delete_configuration_profile() (OVF method), 111
 delete_configuration_profile() (VMDescription method), 42
 delete_item() (OVFHardware method), 120
 description (COTAddDisk attribute), 54
 descriptions (COTEditProperties attribute), 67
 destroy() (COTGenericSubmodule method), 51
 destroy() (VMDescription method), 42
 detect_type_from_name()
 (COT.vm_description.VMDescription class method), 42
 detect_type_from_name() (OVF static method), 111
 device_address() (in module COT.data_validation), 75
 device_info_str() (OVF method), 111
 disk description, 21
 disk device, 21
 disk drive, 21
 disk element, 21
 disk file, 21
 disk image, 21
 disk item, 21
 disk reference, 21
 disk_format (DiskRepresentation attribute), 130
 disk_format (ISO attribute), 131
 disk_format (QCOW2 attribute), 131
 disk_format (RAW attribute), 132
 disk_format (VMDK attribute), 132
 disk_image (COTAddDisk attribute), 54
 disk_representation_from_file() (in module COT.disks), 132
 disk_subformat (DiskRepresentation attribute), 130
 disk_subformat (ISO attribute), 131
 disk_subformat (VMDK attribute), 132
 diskname (COTAddDisk attribute), 54
 DiskRepresentation (class in COT.disks.disk), 129
 download_and_expand_tgz() (Helper static method), 99
 drive_type (COTAddDisk attribute), 54

E

edit_properties_interactive() (COTEditProperties method), 66
 ELEMENT_KEY_SUFFIX (OVFItem attribute), 126
 environment_properties (OVF attribute), 117
 environment_properties (VMDescription attribute), 47
 environment_transports (OVF attribute), 117
 environment_transports (VMDescription attribute), 47
 exists (FileInTAR attribute), 80
 exists (FileOnDisk attribute), 81
 expand_list_wildcard() (in module COT.edit_hardware), 64
 extra_files (COTInjectConfig attribute), 69

F

factor_bytes() (in module COT.ovf.ovf), 119

FatDisk (class in COT.helpers.fatdisk), 104
file (COTAddFile attribute), 58
file element, 21
file reference, 21
file_checksum() (in module COT.data_validation), 75
file_id (COTAddDisk attribute), 54
file_id (COTAddFile attribute), 58
file_id (COTRemoveFile attribute), 71
file_is_this_type() (COT.disks.disk.DiskRepresentation class method), 130
file_is_this_type() (COT.disks.iso.ISO class method), 130
file_path (COTRemoveFile attribute), 71
FileInTAR (class in COT.file_reference), 79
FileOnDisk (class in COT.file_reference), 80
files (DiskRepresentation attribute), 130
files (ISO attribute), 131
files (RAW attribute), 132
fill_examples() (CLI method), 94
fill_examples() (UI method), 92
fill_usage() (CLI method), 95
fill_usage() (UI method), 92
find_all_children() (COT.xml_file.XML class method), 49
find_all_items() (OVFHardware method), 120
find_child() (COT.xml_file.XML class method), 49
find_device_location() (OVF method), 111
find_device_location() (VMDescription method), 42
find_disk_from_file_id() (OVF method), 111
find_empty_drive() (OVF method), 111
find_empty_drive() (VMDescription method), 42
find_item() (OVFHardware method), 120
find_item_from_disk() (OVF method), 111
find_item_from_file() (OVF method), 112
find_open_controller() (OVF method), 112
find_open_controller() (VMDescription method), 42
find_parent_from_item() (OVF method), 112
find_unused_instance_id() (OVFHardware method), 121
finished() (COTGenericSubmodule method), 51
finished() (COTSubmodule method), 52
fixup_ovftool_args() (COTDeployESXi method), 61
fixup_serial_ports() (COTDeployESXi method), 61
force (UI attribute), 93
formatter() (in module COT.cli), 96
from_cli_string() (COT.deploy.SerialConnection class method), 59
from_other_image() (COT.disks.disk.DiskRepresentation class method), 130
from_other_image() (COT.disks.iso.ISO class method), 130
from_other_image() (COT.disks.qcow2.QCOW2 class method), 131
from_other_image() (COT.disks.raw.RAW class method), 131
from_other_image() (COT.disks.vmdk.VMDK class method), 132
full_version (COTEditProduct attribute), 66

G

GCC (class in COT.helpers.gcc), 104
generate_items() (OVFItem method), 124
generate_manifest() (OVF method), 112
generic_parser (COTDeploy attribute), 59
GenericPlatform (class in COT.platforms.generic), 81
GenISOImage (class in COT.helpers.mkisofs), 105
get() (OVFItem method), 124
get_all_values() (OVFItem method), 124
get_capacity_from_disk() (OVF method), 112
get_common_subtype() (OVF method), 112
get_common_subtype() (VMDescription method), 42
get_file_ref_from_disk() (OVF method), 112
get_file_ref_from_disk() (VMDescription method), 42
get_id_from_disk() (OVF method), 112
get_id_from_disk() (VMDescription method), 43
get_id_from_file() (OVF method), 112
get_id_from_file() (VMDescription method), 43
get_input() (CLI method), 95
get_input() (UI method), 92
get_item_count() (OVFHardware method), 121
get_item_count_per_profile() (OVFHardware method), 121
get_nic_count() (OVF method), 112
get_nic_count() (VMDescription method), 43
get_nonintersecting_set_list() (OVFItem method), 124
get_ns() (XML static method), 49
get_object_from_connection() (in module COT.deploy_esxi), 62
get_password() (CLI method), 95
get_password() (UI method), 93
get_path_from_file() (OVF method), 113
get_path_from_file() (VMDescription method), 43
get_property_value() (OVF method), 113
get_property_value() (VMDescription method), 43
get_serial_connectivity() (OVF method), 113
get_serial_connectivity() (VMDescription method), 43
get_serial_count() (OVF method), 113
get_serial_count() (VMDescription method), 43
get_value() (OVFItem method), 124
guess_controller_type() (in module COT.add_disk), 55
guess_drive_type_from_extension() (in module COT.add_disk), 55
guess_list_wildcard() (in module COT.edit_hardware), 65
guess_manpath() (in module COT.install_helpers), 70
guess_nic_name() (COT.platforms.cisco_csr1000v.CSR1000V class method), 83
guess_nic_name() (COT.platforms.cisco_iosv.IOSv class method), 84

guess_nic_name() (COT.platforms.cisco_iosxrv.IOSXRV class method), 86
 guess_nic_name() (COT.platforms.cisco_iosxrv.IOSXRVLC instance_id (OVFItem attribute), 126
 class method), 87
 guess_nic_name() (COT.platforms.cisco_iosxrv.IOSXRVLP (class in COT.platforms.cisco_iosrv), 84
 class method), 87
 guess_nic_name() (COT.platforms.cisco_iosxrv_9000.IOSXRV9000 (class in COT.platforms.cisco_iosxrv_9000), 88
 class method), 88
 guess_nic_name() (COT.platforms.cisco_nexus_9000v.Nexus9000vLC (class in COT.platforms.cisco_iosxrv), 87
 class method), 88
 guess_nic_name() (COT.platforms.cisco_nxosv.NXOSv is_known_product_class() (in module COT.platforms), 91
 class method), 90
 guess_nic_name() (COT.platforms.generic.GenericPlatformISOInfo (class in COT.helpers.isoinfo), 104
 class method), 82
 item_match() (OVFHardware method), 121
 item_tag_for_namespace() (OVFNameHelper1 method), 127

H

hardware controller, 21
 hardware element, 21
 hardware item, 21
 hardware_subtype (OVFItem attribute), 126
 hardware_type (OVFItem attribute), 126
 has_profile() (OVFItem method), 124
 Helper (class in COT.helpers.helper), 98
 helper_select() (in module COT.helpers.helper), 103
 HelperDict (class in COT.helpers.helper), 100
 HelperError, 98
 HelperNotFoundError, 98
 helpers (in module COT.helpers.helper), 103
 host (COTDeployESXi attribute), 61
 hypervisor (COTDeploy attribute), 59

I

ide_subtype (COTEditHardware attribute), 63
 ide_subtypes (COTEditHardware attribute), 63
 info_string() (OVF method), 113
 info_string() (VMDescription method), 43
 INFO_STRING_DISK_COLUMNS_WIDTH (OVF attribute), 117
 INFO_STRING_DISK_TEMPLATE (OVF attribute), 117
 INFO_STRING_FILE_TEMPLATE (OVF attribute), 117
 info_uri (Helper attribute), 100
 input_file (VMDescription attribute), 47
 install() (Helper method), 100
 install_helper() (COTInstallHelpers method), 70
 install_manpages() (in module COT.install_helpers), 70
 install_package() (AptGet method), 103
 install_package() (Brew method), 103
 install_package() (PackageManager method), 101
 install_package() (Port method), 106
 install_package() (Yum method), 107
 installable (FatDisk attribute), 104
 installable (Helper attribute), 100
 installable (OVFTool attribute), 106

K

kind (SerialConnection attribute), 60

L

labels (COTEditProperties attribute), 67
 list_union() (in module COT.ovf.item), 126
 LITERAL_CLI_STRING (CSR1000V attribute), 84
 LITERAL_CLI_STRING (GenericPlatform attribute), 83
 LITERAL_CLI_STRING (IOSv attribute), 85
 LITERAL_CLI_STRING (IOSXRV attribute), 87
 LITERAL_CLI_STRING (Nexus9000v attribute), 89
 LITERAL_CLI_STRING (NXOSv attribute), 90
 locator (COTDeployESXi attribute), 61

M

mac_address() (in module COT.data_validation), 76
 mac_addresses_list (COTEditHardware attribute), 63
 main() (CLI method), 96
 main() (in module COT.cli), 97
 Make (class in COT.helpers.make), 104
 manpages_helper() (COTInstallHelpers method), 70
 match_or_die() (in module COT.data_validation), 76
 memory (COTEditHardware attribute), 63
 MEMORY_REGEXP (COTEditHardware attribute), 63
 mkdir() (Helper static method), 100
 MkISOFS (class in COT.helpers.mkisofs), 105

N

name (Helper attribute), 100
 name_helper() (in module COT.ovf.name_helper), 128
 namespace_for_item_tag() (OVFNameHelper1 method), 127
 namespace_for_resource_type() (OVFNameHelper1 method), 127
 natural_sort() (in module COT.data_validation), 76
 network_descriptions (COTEditHardware attribute), 63
 network_descriptions (OVF attribute), 117

network_descriptions (VMDescription attribute), 47
network_map (COTDeploy attribute), 59
networks (OVF attribute), 118
networks (VMDescription attribute), 47
new_item() (OVFHardware method), 121
Nexus9000v (class in COT.platforms.cisco_nexus_9000v), 88
nic_names (COTEditHardware attribute), 64
nic_networks (COTEditHardware attribute), 64
nic_type (COTEditHardware attribute), 64
nic_types (COTEditHardware attribute), 64
NIC_TYPES (in module COT.data_validation), 79
nics (COTEditHardware attribute), 64
no_whitespace() (in module COT.data_validation), 77
non_negative_int() (in module COT.data_validation), 77
NSM (OVFNameHelper0 attribute), 127
NSM (OVFNameHelper1 attribute), 127
NSM (OVFNameHelper2 attribute), 128
NXOSv (class in COT.platforms.cisco_nxosv), 90

O

open() (FileInTAR method), 80
open() (FileOnDisk method), 81
options (SerialConnection attribute), 60
output (COTSubmodule attribute), 52
output_file (OVF attribute), 118
output_file (VMDescription attribute), 47
OVF, 21
OVF (class in COT.ovf.ovf), 108
OVF descriptor, 21
ovf_version (OVF attribute), 118
OVFHardware (class in COT.ovf.hardware), 120
OVFHardwareDataError, 120
OVFItem (class in COT.ovf.item), 123
OVFItemDataError, 123
OVFNameHelper0 (class in COT.ovf.name_helper), 127
OVFNameHelper1 (class in COT.ovf.name_helper), 127
OVFNameHelper2 (class in COT.ovf.name_helper), 127
OVFTool (class in COT.helpers.ovftool), 105
ovftool_args (COTDeployESXi attribute), 62

P

package (COTReadOnlySubmodule attribute), 52
package (COTSubmodule attribute), 53
package_list (COTInfo attribute), 68
PackageManager (class in COT.helpers.helper), 101
parse_args() (CLI method), 96
password (COTDeploy attribute), 59
path (DiskRepresentation attribute), 130
path (Helper attribute), 100
platform (OVF attribute), 118
platform (VMDescription attribute), 47
platform_from_product_class() (in module COT.platforms), 91

PLATFORM_NAME (CSR1000V attribute), 84
PLATFORM_NAME (GenericPlatform attribute), 83
PLATFORM_NAME (IOSv attribute), 85
PLATFORM_NAME (IOSXRv attribute), 87
PLATFORM_NAME (IOSXRv9000 attribute), 88
PLATFORM_NAME (IOSXRvLC attribute), 87
PLATFORM_NAME (IOSXRvRP attribute), 87
PLATFORM_NAME (Nexus9000v attribute), 89
PLATFORM_NAME (NXOSv attribute), 90
Port (class in COT.helpers.port), 106
positive_int() (in module COT.data_validation), 77
power_on (COTDeploy attribute), 59
product (COTEditProduct attribute), 66
product (OVF attribute), 118
product_class (COTEditProduct attribute), 66
product_class (OVF attribute), 118
product_class (VMDescription attribute), 47
product_url (COTEditProduct attribute), 66
product_url (OVF attribute), 118
profile_info_list() (OVF method), 113
profile_info_string() (OVF method), 113
profile_info_string() (VMDescription method), 43
PROFILE_INFO_TEMPLATE (OVF attribute), 117
profiles (COTEditHardware attribute), 64
properties (COTEditProperties attribute), 67
properties (OVFItem attribute), 126
property_names (OVFItem attribute), 126
property_profiles() (OVFItem method), 124
property_values() (OVFItem method), 124
PyVmomiVMReconfigSpec (class in COT.deploy_esxi), 62

Q

QCOW2 (class in COT.disks.qcow2), 131
QEMUImg (class in COT.helpers.qemu_img), 106

R

RAW (class in COT.disks.raw), 131
ready_to_run() (COTAddDisk method), 53
ready_to_run() (COTAddFile method), 57
ready_to_run() (COTDeploy method), 58
ready_to_run() (COTDeployESXi method), 61
ready_to_run() (COTEditHardware method), 63
ready_to_run() (COTEditProduct method), 65
ready_to_run() (COTEditProperties method), 66
ready_to_run() (COTGenericSubmodule method), 51
ready_to_run() (COTInfo method), 68
ready_to_run() (COTInjectConfig method), 69
ready_to_run() (COTReadOnlySubmodule method), 52
ready_to_run() (COTRemoveFile method), 71
ready_to_run() (COTSubmodule method), 52
register_namespace() (in module COT.xml_file), 50
remove_file() (OVF method), 113
remove_file() (VMDescription method), 44

remove_profile() (OVFItem method), 125
 RES_MAP (OVFNameHelper1 attribute), 127
 root (XML attribute), 50
 run() (CLI method), 96
 run() (COTAddDisk method), 53
 run() (COTAddFile method), 58
 run() (COTDeploy method), 58
 run() (COTDeployESXi method), 61
 run() (COTEditHardware method), 63
 run() (COTEditProduct method), 66
 run() (COTEditProperties method), 67
 run() (COTGenericSubmodule method), 52
 run() (COTHelp method), 68
 run() (COTInfo method), 68
 run() (COTInjectConfig method), 69
 run() (COTInstallHelpers method), 70
 run() (COTRemoveFile method), 71
 run() (COTSubmodule method), 52

S

scsi_subtype (COTEditHardware attribute), 64
 scsi_subtypes (COTEditHardware attribute), 64
 search_for_elements() (in module COT.add_disk), 56
 search_from_controller() (OVF method), 114
 search_from_controller() (VMDescription method), 44
 search_from_file_id() (OVF method), 114
 search_from_file_id() (VMDescription method), 44
 search_from_filename() (OVF method), 114
 search_from_filename() (VMDescription method), 44
 secondary_config_file (COTInjectConfig attribute), 69
 SECONDARY_CONFIG_TEXT_FILE (GenericPlatform attribute), 83
 SECONDARY_CONFIG_TEXT_FILE (IOSXRv attribute), 87
 SECONDARY_CONFIG_TEXT_FILE (IOSXRvLC attribute), 87
 serial_connection (COTDeploy attribute), 59
 serial_connection (COTDeployESXi attribute), 62
 serial_connectivity (COTEditHardware attribute), 64
 serial_ports (COTEditHardware attribute), 64
 SerialConnection (class in COT.deploy), 59
 server (COTDeployESXi attribute), 62
 set_capacity_of_disk() (OVF method), 114
 set_cpu_count() (OVF method), 115
 set_cpu_count() (VMDescription method), 44
 set_ide_subtype() (VMDescription method), 44
 set_ide_subtypes() (OVF method), 115
 set_ide_subtypes() (VMDescription method), 44
 set_instance_attributes() (CLI static method), 96
 set_item_count_per_profile() (OVFHardware method), 121
 set_item_values_per_profile() (OVFHardware method), 122
 set_memory() (OVF method), 115

set_memory() (VMDescription method), 45
 set_nic_count() (OVF method), 115
 set_nic_count() (VMDescription method), 45
 set_nic_mac_addresses() (OVF method), 115
 set_nic_mac_addresses() (VMDescription method), 45
 set_nic_names() (OVF method), 115
 set_nic_names() (VMDescription method), 45
 set_nic_networks() (OVF method), 115
 set_nic_networks() (VMDescription method), 45
 set_nic_type() (VMDescription method), 45
 set_nic_types() (OVF method), 116
 set_nic_types() (VMDescription method), 46
 set_or_make_child() (COT.xml_file.XML class method), 50
 set_property() (OVFItem method), 125
 set_property_value() (OVF method), 116
 set_property_value() (VMDescription method), 46
 set_scsi_subtype() (VMDescription method), 46
 set_scsi_subtypes() (OVF method), 116
 set_scsi_subtypes() (VMDescription method), 46
 set_serial_connectivity() (OVF method), 116
 set_serial_connectivity() (VMDescription method), 46
 set_serial_count() (OVF method), 116
 set_serial_count() (VMDescription method), 46
 set_value_for_all_items() (OVFHardware method), 122
 set_verbosity() (CLI method), 96
 size (FileInTAR attribute), 80
 size (FileOnDisk attribute), 81
 SmarterConnection (class in COT.deploy_esxi), 62
 strip_ns() (XML static method), 50
 subcommand (COTHelp attribute), 68
 subparsers (COTDeploy attribute), 59
 subtype (COTAddDisk attribute), 54
 SUPPORTED_NIC_TYPES (CSR1000V attribute), 84
 SUPPORTED_NIC_TYPES (GenericPlatform attribute), 83
 SUPPORTED_NIC_TYPES (IOSv attribute), 85
 SUPPORTED_NIC_TYPES (IOSXRv attribute), 87
 SUPPORTED_NIC_TYPES (IOSXRv9000 attribute), 88
 SUPPORTED_NIC_TYPES (Nexus9000v attribute), 89
 SUPPORTED_NIC_TYPES (NXOSv attribute), 90
 system_types (OVF attribute), 118
 system_types (VMDescription attribute), 47

T

tar() (OVF method), 117
 terminal_width (CLI attribute), 96
 terminal_width (UI attribute), 93
 to_string() (in module COT.data_validation), 78
 transports (COTEditProperties attribute), 67
 tree (XML attribute), 50
 truth_value() (in module COT.data_validation), 78

U

UI (class in COT.ui_shared), 91
ui (COTGenericSubmodule attribute), 52
unsure_how_to_install() (Helper method), 100
unsure_how_to_install() (OVFTool method), 106
untar() (OVF method), 117
unwrap_connection_error() (SmarterConnection static method), 62
update_existing_item_count_per_profile() (OVFHardware method), 122
update_xml() (OVFHardware method), 122
user_configurable (COTEditProperties attribute), 67
USER_INTERFACE (Helper attribute), 100
username (COTDeploy attribute), 59

V

validate() (OVFItem method), 125
validate_and_update_file_references() (OVF method), 117
validate_and_update_networks() (OVF method), 117
validate_controller_address() (in module COT.add_disk), 56
validate_cpu_count() (COT.platforms.cisco_csr1000v.CSR1000V class method), 84
validate_cpu_count() (COT.platforms.cisco_iosv.IOSv class method), 85
validate_cpu_count() (COT.platforms.cisco_iosxrv.IOSXrv class method), 86
validate_cpu_count() (COT.platforms.cisco_iosxrv_9000.IOSXrv9000 class method), 88
validate_cpu_count() (COT.platforms.cisco_nexus_9000v.Nexus9000v class method), 89
validate_cpu_count() (COT.platforms.cisco_nxosv.NXOSv class method), 90
validate_cpu_count() (COT.platforms.generic.GenericPlatform class method), 82
validate_elements() (in module COT.add_disk), 57
validate_hardware() (OVF method), 117
validate_hardware() (VMDescription method), 47
validate_int() (in module COT.data_validation), 79
validate_kind() (COT.deploy.SerialConnection class method), 60
validate_memory_amount() (COT.platforms.cisco_csr1000v.CSR1000V class method), 84
validate_memory_amount() (COT.platforms.cisco_iosv.IOSv class method), 85
validate_memory_amount() (COT.platforms.cisco_iosxrv.IOSXrv class method), 86
validate_memory_amount() (COT.platforms.cisco_iosxrv_9000.IOSXrv9000 class method), 88
validate_memory_amount() (COT.platforms.cisco_nexus_9000v.Nexus9000v class method), 89
validate_memory_amount() (COT.platforms.cisco_nxosv.NXOSv class method), 90
validate_memory_amount() (COT.platforms.generic.GenericPlatform class method), 82
validate_nic_count() (COT.platforms.cisco_csr1000v.CSR1000V class method), 84
validate_nic_count() (COT.platforms.cisco_iosv.IOSv class method), 85
validate_nic_count() (COT.platforms.cisco_iosxrv.IOSXrv class method), 86
validate_nic_count() (COT.platforms.cisco_iosxrv.IOSXrvRP class method), 87
validate_nic_count() (COT.platforms.cisco_iosxrv_9000.IOSXrv9000 class method), 88
validate_nic_count() (COT.platforms.cisco_nexus_9000v.Nexus9000v class method), 89
validate_nic_count() (COT.platforms.generic.GenericPlatform class method), 82
validate_nic_type() (COT.platforms.generic.GenericPlatform class method), 82
validate_nic_types() (COT.platforms.generic.GenericPlatform class method), 83
validate_options() (COT.deploy.SerialConnection class method), 60
validate_serial_count() (COT.platforms.cisco_csr1000v.CSR1000V class method), 84
validate_serial_count() (COT.platforms.cisco_iosv.IOSv class method), 85
validate_serial_count() (COT.platforms.cisco_iosxrv.IOSXrv class method), 86
validate_serial_count() (COT.platforms.cisco_iosxrv.IOSXrvLC class method), 87
validate_serial_count() (COT.platforms.cisco_nexus_9000v.Nexus9000v class method), 89
validate_serial_count() (COT.platforms.cisco_nxosv.NXOSv class method), 90
validate_serial_count() (COT.platforms.generic.GenericPlatform class method), 83
validate_value() (COT.deploy.SerialConnection class method), 60
value (SerialConnection attribute), 60
value_add_wildcards() (OVFItem method), 125
value_replace_wildcards() (OVFItem method), 125
ValueMismatchError, 72
ValueTooHighError, 72
ValueTooLowError, 72
ValueUnsupportedError, 72
vendor (COTEditProduct attribute), 66
vendor (OVF attribute), 118

vendor_url (COTEditProduct attribute), 66
vendor_url (OVF attribute), 118
verbosity (COTInfo attribute), 69
verbosity_options (VMDescription attribute), 47
verify_manpages() (in module COT.install_helpers), 70
version (COTEditProduct attribute), 66
version (Helper attribute), 100
version_long (OVF attribute), 118
version_long (VMDescription attribute), 47
version_short (OVF attribute), 118
version_short (VMDescription attribute), 47
virtual_system_type (COTEditHardware attribute), 64
vm (COTGenericSubmodule attribute), 52
vm_name (COTDeploy attribute), 59
VMContextManager (class in COT.vm_context_manager), 48
VMDescription (class in COT.vm_description), 39
VMDK (class in COT.disks.vmdk), 132
VMDKTool (class in COT.helpers.vmdktool), 106
VMFactory (class in COT.vm_factory), 48
VMInitError, 39

W

write() (OVF method), 117
write() (VMDescription method), 47
write_xml() (XML method), 50

X

XML (class in COT.xml_file), 48
xml_reindent() (XML method), 50
XorrISO (class in COT.helpers.mkisofs), 105

Y

Yum (class in COT.helpers.yum), 107