common-ovf-tool Documentation

Release 1.3.2

the COT project developers

April 09, 2015

Contents

CHAPTER 1

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.
- Deploy an OVF/OVA to an ESXi (VMware vSphere or vCenter) server to provision a new virtual machine (VM).

1.1 Examples

Displaying a summary of OVA contents:

```
> cot info --brief iosxrv.5.1.1.ova
--------------------------------------------------------------------------------
iosxrv.5.1.1.ova
COT detected platform type: Cisco IOS XRv
--------------------------------------------------------------------------------
Product:  Cisco IOS XRv
Vendor:   Cisco Systems, Inc.
Version:  5.1.1

Files and Disks:     File Size  Capacity Device
                     --------- --------- --------------------
  iosxrv.vmdk        271.59 MB   3.00 GB harddisk @ IDE 0:0

Hardware Variants:
  System types:             vmx-08 Cisco:Internal:VMCloud-01
  Ethernet device types:    E1000

Configuration Profiles:   CPUs    Memory NICs Serials Disks/Capacity
                          ---- --------- ---- ------- --------------
  1CPU-3GB-2NIC (default)    1   3.00 GB    2       2  1 /   3.00 GB
```

```
  2CPU-4GB-8NIC              2   4.00 GB    8       2  1 /   3.00 GB
  4CPU-6GB-10NIC             4   6.00 GB   10       2  1 /   3.00 GB
```

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
```

CHAPTER 2

Installing COT

2.1 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.

On Mac OS X you can install COT from MacPorts, or you can install it manually by following the same steps as Linux, described below.

2.2 Download 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 zxf cot.tgz
cd cot-master
```

or

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

2.3 Install the COT libraries and script

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

2.4 Install 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`) to create ISO (CD-ROM) images for platforms that use ISOs.
- 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.

CHAPTER 3

Using COT

3.1 Getting CLI help

3.1.1 Synopsis

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

3.1.2 Description

Common OVF Tool (COT), version 1.3.2+0.g932bc66.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>`.

3.1.3 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 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

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

3.2.1 Synopsis

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

3.2.2 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/)

3.2.3 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.

3.2.4 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 now
    INFO: Calling 'sudo cp fatdisk /usr/local/bin/fatdisk'...
    INFO: ...done
    INFO: Successfully installed 'fatdisk'
    INFO: Installing 'vmdktool'...
    INFO: vmdktool requires 'zlib'... installing 'zlib'
    INFO: Calling 'sudo apt-get -q install zlib1g-dev'...
(...)
    INFO: ...done
    INFO: Compiling 'vmdktool'
    INFO: Calling 'make CFLAGS="-D_GNU_SOURCE -g -O -pipe"'...
(...)
    INFO: ...done
    INFO: Compilation complete, installing now.
    INFO: Calling 'sudo mkdir -p --mode=755 /usr/local/man/man8'...
    INFO: ...done
    INFO: Calling 'sudo make install'...
install -s vmdktool /usr/local/bin/
install vmdktool.8 /usr/local/man/man8/
    INFO: ...done
    INFO: Successfully installed 'vmdktool'
    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.

3.3 Inspecting OVF contents with `cot info`

3.3.1 Synopsis

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

3.3.2 Description

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

3.3.3 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)

3.4 Updating version information with `cot edit-product`

3.4.1 Synopsis

```
cot edit-product --help
cot <opts> edit-product PACKAGE [-o OUTPUT] [-v SHORT_VERSION]
                        [-V FULL_VERSION]
```

3.4.2 Description

Edit product information attributes of the given OVF or OVA

3.4.3 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

-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"

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

3.5.1 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]
```

3.5.2 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.

3.5.3 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 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 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)

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

3.6.1 Synopsis

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

3.6.2 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.

3.6.3 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)

3.7 Customizing hardware profiles with `cot edit-hardware`

3.7.1 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 ...]] [-c CPUS]
                         [-m MEMORY] [-n NICS]
                         [--nic-type {e1000,virtio,vmxnet3}]
                         [-N NETWORK [NETWORK2 ...]]
                         [-M MAC1 [MAC2 ...]]
                         [--nic-names NAME1 [NAME2 ...]]
                         [-s SERIAL_PORTS] [-S URI1 [URI2 ...]]
                         [--scsi-subtype SCSI_SUBTYPE]
                         [--ide-subtype IDE_SUBTYPE]
```

3.7.2 Description

Edit hardware properties of the specified OVF or OVA

3.7.3 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)

Computational hardware options

-c CPUS, --cpus CPUS Set the number of CPUs.

-m MEMORY, --memory MEMORY Set the amount of RAM. (Examples: "4096MB", "4GB")

Network interface options

-n NICS, --nics NICS Set the number of NICs.

--nic-type <e1000,virtio,vmxnet3> Set the hardware type for all NICs. (default: do not change existing NICs, and new NICs added will match the existing type.)

-N <NETWORK...>, --nic-networks <NETWORK...> Specify a series of one or more network names to map NICs to. If N network names are specified, the first (N-1) NICs will be mapped to the first (N-1) networks and all remaining NICs will be mapped to the Nth network.

-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

--nic-names <NAME1...> Specify a list of one or more NIC names or patterns to apply to NIC devices. If N names/patterns are specified, the first (N-1) NICs will receive the first (N-1) names and remaining NICs will be named based on the name or pattern of the Nth item. See examples.

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-subtype SCSI_SUBTYPE Set resource subtype (such as "lsilogic" or "virtio") for all SCSI controllers. If an empty string is provided, any existing subtype will be removed.

--ide-subtype IDE_SUBTYPE Set resource subtype (such as "virtio") for all IDE controllers. If an empty string is provided, any existing subtype will be removed.

3.7.4 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
```

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}"
```

Rename the NICs in the output OVA as 'Ethernet0/10', 'Ethernet0/11', 'Ethernet0/12', etc.

```
cot edit-hardware input.ova -o output.ova \
    --nic-names "Ethernet0/{10}"
```

3.8 Customizing OVF environment settings with `cot edit-properties`

3.8.1 Synopsis

```
cot edit-properties --help
cot <opts> edit-properties PACKAGE -p KEY1=VALUE1 [KEY2=VALUE2 ...]
                           [-o OUTPUT]
cot <opts> edit-properties PACKAGE -c CONFIG_FILE [-o OUTPUT]
cot <opts> edit-properties PACKAGE [-o OUTPUT]
```

3.8.2 Description

Configure environment properties of the given OVF or OVA. The user may specify key-value pairs as command-line arguments or may provide a config-file to read from. If neither are specified, the program will run interactively.

3.8.3 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

-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...>, --properties <KEY1=VALUE1...> Set the given property key-value pair(s). This argument may be repeated as needed to specify multiple properties to edit.

3.9 Embedding bootstrap configuration with `cot inject-config`

3.9.1 Synopsis

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

3.9.2 Description

Add one or more "bootstrap" configuration file(s) to the given OVF or OVA.

3.9.3 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 Primary configuration text file to embed

-s SECONDARY_CONFIG_FILE, --secondary-config-file SECONDARY_CONFIG_FILE Secondary configuration text file to embed (currently only supported in IOS XRv for admin config)

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

3.10.1 Synopsis

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

3.10.2 Description

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

3.10.3 Options

PACKAGE OVF descriptor or OVA file

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

Hypervisors

esxi Deploy to ESXi, vSphere, or vCenter

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

3.11.1 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 ...]]
                               [-d DATASTORE] [-o=OVFTOOL_ARGS]
```

3.11.2 Description

Deploy OVF/OVA to ESXi/vCenter/vSphere hypervisor

3.11.3 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.

-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"

3.11.4 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
```

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"
```

CHAPTER 4

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

4.1 Follow coding guidelines

4.1.1 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.

4.1.2 Coding style

To verify that your code meets applicable Python coding standards, use flake8 (`pip install flake8`):

```
cot/$ flake8
./COT/tests/ovf.py:180:80: E501 line too long (80 > 79 characters)
./COT/tests/ovf.py:184:77: F841 local variable 'ovf' is assigned to but never used
./COT/tests/ovf.py:184:80: E501 line too long (80 > 79 characters)
./COT/tests/ovf.py:196:40: F841 local variable 'ova' is assigned to but never used
./COT/tests/ovf.py:210:75: F841 local variable 'ovf' is assigned to but never used
./COT/ovf.py:776:5: E303 too many blank lines (2)
```

Fix any errors it reports, and run again until no errors are reported.

4.2 Add automated unit tests

Whether adding new functionality or fixing a bug, **please** add appropriate unit test case(s) under `COT/tests/` 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
...
py32 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
...
pypy runtests: commands[0] | coverage run --append setup.py test --quiet
...
stats runtests: commands[0] | coverage combine
stats runtests: commands[1] | coverage report -i
Name                         Stmts   Miss  Cover
----------------------------------------------
COT/__init__.py                  4      0   100%
COT/add_disk.py                147      1    99%
COT/add_file.py                 50      0   100%
COT/cli.py                     143      6    96%
COT/data_validation.py          69      0   100%
COT/deploy.py                  142      5    96%
COT/edit_hardware.py           159      0   100%
COT/edit_product.py             36      0   100%
COT/edit_properties.py         104     40    62%
COT/helper_tools.py            171      3    98%
COT/info.py                     41      0   100%
COT/inject_config.py            87      2    98%
COT/ovf.py                    1586     52    96%
COT/platforms.py               173      0   100%
COT/submodule.py                80      2    98%
COT/ui_shared.py                24      0   100%
COT/vm_context_manager.py       12      0   100%
COT/vm_description.py          119      1    99%
COT/vm_factory.py               25      0   100%
COT/xml_file.py                112      0   100%
----------------------------------------------
TOTAL                         3284    112    96%
stats runtests: commands[2] | coverage html -i
...
flake8 runtests: commands[0] | flake8
_________________ summary _______________
  clean: commands succeeded
  py26: commands succeeded
  py27: commands succeeded
```

```
py32: commands succeeded
py33: commands succeeded
py34: commands succeeded
pypy: commands succeeded
stats: commands succeeded
flake8: commands succeeded
congratulations :)
```

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

4.3 Update documentation

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

4.4 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.

CHAPTER 5

Credits

We would like to thank:

- For evangelization, user feedback and bug reports:
  - Mark Coverdill
  - Jonathan Muslow
  - Rick Ogg
  - David Rosenfeld
  - Perumal Venkatesh
- 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

CHAPTER 6

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.

6.1 Virtual machine definition modules

COT.vm_description	Abstract superclass for reading, editing, and writing VMs.
COT.vm_factory	Factory for virtual machine objects.
COT.vm_context_manager	Context manager for virtual machine definitions.
COT.xml_file	Reading, editing, and writing XML files.
COT.ovf	Module for handling OVF and OVA virtual machine description files.

6.1.1 COT.vm_description module

Abstract superclass for reading, editing, and writing VMs.

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

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

Continued on next page

Table 6.3 – continued from previous page

environment_properties	The array of environment properties.
networks	The list of network names currently defined in this VM.
system_types	List of virtual system type(s) supported by this virtual machine.
version_short	A short string describing the product version.
version_long	A long string describing the product version.

add_controller_device (*type*, *subtype*, *address*, *ctrl_item=None*)
Create a new IDE or SCSI controller, or update existing one.

Parameters

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

Returns New or updated controller device object

add_disk (*file_path*, *file_id*, *disk_type*, *disk=None*)
Add a new disk object to the VM or overwrite the provided one.

Parameters

- **file_path** (*str*) – Path to disk image file
- **file_id** (*str*) – Identifier string for the file/disk mapping
- **disk_type** (*str*) – 'harddisk' or 'cdrom'
- **disk** – Existing disk object to overwrite

Returns New or updated disk object

add_disk_device (*type*, *address*, *name*, *description*, *disk*, *file*, *ctrl_item*, *disk_item=None*)
Add a new disk device to the VM or update the provided one.

Parameters

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

Returns New or updated disk device object.

add_file (*file_path*, *file_id*, *file=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** – Existing file object to overwrite
- **disk** – Existing disk object referencing `file`.

Returns New or updated file object

check_sanity_of_disk_device(*disk*, *file*, *disk_item*, *ctrl_item*)
Check if the given disk is linked properly to the other objects.

Parameters

- **disk** – Disk object to validate
- **file** – File object which this disk should be linked to (optional)
- **disk_item** – Disk device object which should link to this disk (optional)
- **ctrl_item** – 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*)
Import each line of a text file into a configuration property.

Parameters **file** (*str*) – File name to import.

convert_disk_if_needed(*file_path*, *kind*)
Convert the disk to a more appropriate format if needed.

Parameters

- **file_path** (*str*) – Image to inspect and possibly convert
- **kind** (*str*) – Image type (harddisk/cdrom)

Returns

- `file_path`, if no conversion was required
- or a file path in `output_dir` containing the converted image

create_configuration_profile(*id*, *label*, *description*)
Create/update a configuration profile with the given ID.

Parameters

- **id** – 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.

Parameters

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

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.

Returns A string representing a recognized and supported type of file

Raises ValueUnsupportedError if we don't know how to handle this file.

find_device_location(*device*)
Find the controller type and address of a given device object.

Parameters **device** – Hardware device object.

Returns `(type, address)`, such as `("ide", "1:0")`.

find_empty_drive(*type*)
Find a disk device that exists but contains no data.

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

Returns Hardware device object, or None.

find_open_controller(*type*)
Find the first open slot on a controller of the given type.

Parameters **type** (*str*) – `'ide'` or `'scsi'`

Returns `(controller_device, address_string)` or `(None, None)`

get_common_subtype(*type*)
Get the sub-type common to all devices of the given type.

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

Returns `None`, if multiple such devices exist and they do not all have the same sub-type.

Returns Subtype string common to all devices of the type.

get_file_ref_from_disk(*disk*)
Get the file reference from the given opaque disk object.

Parameters **disk** – Disk object to query

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

get_id_from_file(*file*)
Get the file ID from the given opaque file object.

Parameters **file** – File object to query

Returns 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.

Return type dict

Returns `{ profile_name : nic_count }`

get_path_from_file(*file*)
Get the file path from the given opaque file object.

Parameters **file** – File object to query

Returns 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 Value of this property, or `None`

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

Return type dict

Returns `{ profile_name : serial_count }`

get_subtype_from_device(*device*)
Get the sub-type of the given opaque device object.

Parameters **device** – Device object to query

Returns `None`, or string such as 'virtio' or 'lsilogic'

get_type_from_device(*device*)
Get the type of the given opaque device object.

Parameters **device** – Device object to query

Returns string such as 'ide' or 'memory'

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 Wrapped, appropriately verbose string.

profile_info_string(*width=79*, *verbosity_option=None*, *enumerate=False*)
Get a string summarizing available configuration profiles.

Parameters

- **TEXT_WIDTH** (*int*) – Line length to wrap to if possible
- **verbosity_option** (*str*) – `'brief'`, `None` (default), or `'verbose'`
- **enumerate** (*boolean*) – If `True`, number the profiles.

Returns Appropriately formatted and verbose string.

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

Returns `(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 `(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 (*type*, *profile_list*)
Set the device subtype for the IDE controller(s).

Parameters

- **type** (*str*) – IDE subtype string
- **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).

Since NICs are often named sequentially, this API supports a wildcard option for the final element in `name_list` which can be expanded to automatically assign sequential NIC names. The syntax for the

wildcard option is `{` followed by a number (indicating the starting index for the name) followed by `}`. Examples:

`["eth{0}"]` Expands to `["eth0", "eth1", "eth2", ...]`

`["mgmt0" "eth{10}"]` Expands to `["mgmt0", "eth10", "eth11", "eth12", ...]`

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` (*type*, *profile_list*)
Set the hardware type for NICs.

Parameters

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

`set_property_value` (*key*, *value*)
Set the value of the given property (converting value if needed).

Parameters

- **key** (*str*) – Property identifier
- **value** – Value to set for this property

Returns the (converted) value that was set.

`set_scsi_subtype` (*type*, *profile_list*)
Set the device subtype for the SCSI controller(s).

Parameters

- **type** (*str*) – SCSI subtype string
- **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

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 Profile name or `None` if none are defined.

environment_properties
The array of environment properties.

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

input_file
Data file to read in.

networks
The list of network names currently defined in this VM.

Return type list[str]

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.

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

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

version_long
A long string describing the product version.

version_short
A short string describing the product version.

6.1.2 COT.vm_factory module

Factory for virtual machine objects.

class VMFactory
Creates a VMDescription instance from a specified input file.

classmethod create(*input_file*, *output_file*)
Create an appropriate VMDescription subclass instance from a file.

Raises

- **VMInitError** – if no appropriate class is identified

- **VMInitError** – if the selected subclass raises a ValueUnsupportedError while loading the file.

Parameters

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

Return type instance of `VMDescription` or appropriate subclass

6.1.3 `COT.vm_context_manager` module

Context manager for virtual machine definitions.

class `VMContextManager`(*input_file*, *output_file*)

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()
```

6.1.4 `COT.xml_file` module

Reading, editing, and writing XML files.

class `XML`

Bases: `object`

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

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 **iff** the unaccounted-for tag is in a known namespace.

classmethod `find_all_children`(*parent*, *tag*, *attrib={}*)

Find all matching child elements 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

Return type list of xml.etree.ElementTree.Element instances

classmethod find_child(*parent*, *tag*, *attrib={}*, *required=False*)
Find the unique child element under the specified parent element.

Raises

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

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

Return type xml.etree.ElementTree.Element

classmethod get_ns(*text*)
Get the namespace prefix from an XML element or attribute name.

read_xml(*xml_file*)
Read the given XML file and store it in memory.

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

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

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

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"

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 New or updated child Element.

Return type xml.etree.ElementTree.Element

classmethod strip_ns (*text*)
Remove a namespace prefix from an XML element or attribute name.

write_xml (*file*)
Write pretty XML out to the given file.

Parameters **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.

6.1.5 COT.ovf module

Module for handling OVF and OVA virtual machine description files.

Functions

byte_count	Convert an OVF-style value + multiplier into decimal byte count.
byte_string	Pretty-print the given bytes value.
factor_bytes	Convert a byte count into OVF-style bytes + multiplier.

Classes and Exceptions

OVF	Representation of the contents of an OVF or OVA.
OVFNameHelper	Helper class for OVF.
OVFHardware	Helper class for OVF.
OVFHardwareDataError	The input data used to construct an OVFHardware is not sane.
OVFItem	Helper class for OVF.
OVFItemDataError	Data to be added to an OVFItem conflicts with existing data.

byte_count (*base_val*, *multiplier*)
Convert an OVF-style value + multiplier into decimal byte count.

Inverse operation of factor_bytes().

```
>>> byte_count("128", "byte * 2^20")
134217728
```

Parameters

- **base_val** (*str*) – Base value string (value of ovf:capacity, etc.)
- **multiplier** (*str*) – Multiplier string (value of ovf:capacityAllocationUnits, etc.)

Returns Number of bytes

Return type int

byte_string (*byte_count*, *base_shift=0*)
Pretty-print the given bytes value.

Parameters

- **byte_count** (*float*) – Value
- **base_shift** (*int*) – Base value of byte_count (0 = bytes, 1 = kB, 2 = MB, etc.)

Returns Pretty-printed byte string such as "1.00 GB"

factor_bytes(*byte_count*)

Convert a byte count into OVF-style bytes + multiplier.

Inverse operation of `byte_count()`

```
>>> factor_bytes(134217728)
('128', 'byte * 2^20')
>>> factor_bytes(134217729)
('134217729', 'byte')
```

Parameters **byte_count** (*int*) – Number of bytes

Returns `(base_val, multiplier)`

class **OVF**(*input_file*, *output_file*)

Bases: `COT.vm_description.VMDescription`, `COT.xml_file.XML`

Representation of the contents of an OVF or OVA.

Variables **ovf_version** – Float representing the OVF specification version in use. Supported values at present are 0.9, 1.0, and 2.0.

Properties

`input_file`	Data file to read in.
`output_file`	OVF or OVA file that will be created or updated by `write()`.
`platform`	The platform type, as determined from the OVF descriptor.
`config_profiles`	The list of supported configuration profiles.
`default_config_profile`	The name of the default configuration profile.
`environment_properties`	The array of environment properties.
`networks`	The list of network names currently defined in this VM.
`system_types`	List of virtual system type(s) supported by this virtual machine.
`version_short`	Short descriptive version string (XML `Version` element).
`version_long`	Long descriptive version string (XML `FullVersion` element).

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

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

Parameters

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

Returns New or updated controller device object

add_disk(*file_path*, *file_id*, *disk_type*, *disk=None*)

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

Parameters

- **file_path** (*str*) – Path to disk image file
- **file_id** (*str*) – Identifier string for the file/disk mapping
- **disk_type** (*str*) – 'harddisk' or 'cdrom'
- **disk** – Existing disk object to overwrite

Returns New or updated disk object

add_disk_device (*type*, *address*, *name*, *description*, *disk*, *file*, *ctrl_item*, *disk_item=None*)
Create a new disk hardware device or overwrite an existing one.

Parameters

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

Returns New or updated disk device object.

add_file (*file_path*, *file_id*, *file=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** – Existing file object to overwrite
- **disk** – Existing disk object referencing `file`.

Returns New or updated file object

check_sanity_of_disk_device (*disk*, *file*, *disk_item*, *ctrl_item*)
Check if the given disk is linked properly to the other objects.

Parameters

- **disk** – Disk object to validate
- **file** – File object which this disk should be linked to (optional)
- **disk_item** – Disk device object which should link to this disk (optional)
- **ctrl_item** – 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*)
Import each line of a text file into a configuration property.

Raises NotImplementedError if the `platform` for this OVF does not define `LITERAL_CLI_STRING`

Parameters file (*str*) – File name to import.

convert_disk_if_needed (*file_path*, *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

- **file_path** (*str*) – Image to inspect and possibly convert
- **kind** (*str*) – Image type (harddisk/cdrom)

Returns

- `file_path`, if no conversion was required
- or a file path in `output_dir` containing the converted image

create_configuration_profile (*id*, *label*, *description*)
Create or update a configuration profile with the given ID.

Parameters

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

create_envelope_section_if_absent (*section_tag*, *info_string*, *attrib={}*)
If the OVF doesn't already have the given Section, create it.

Parameters

- **section_tag** (*str*) – XML tag of the desired section.
- **info_string** (*str*) – Info string to set if a new Section is created.
- **attrib** (*dict*) – Attributes to filter by when looking for any existing section.

Returns Section element that was found or created

create_network (*label*, *description*)
Define a new network with the given label and description.

Parameters

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

classmethod 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.

Returns '.ovf' 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 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** – Hardware device object.

Returns `(type, address)`, such as `("ide", "1:0")`.

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 Disk element matching the file, or None

find_empty_drive(*type*)
Find a disk device that exists but contains no data.

Parameters **type** (*str*) – Either 'cdrom' or 'harddisk'

Returns Hardware device object, or None.

find_item_from_disk(*disk*)
Find the disk Item that references the given Disk.

Parameters **disk** (*xml.etree.ElementTree.Element*) – Disk element

Returns `OVFItem` instance, or None

find_item_from_file(*file*)
Find the disk Item that references the given File.

Parameters **file** (*xml.etree.ElementTree.Element*) – File element

Returns `OVFItem` instance, or None.

find_open_controller(*type*)
Find the first open slot on a controller of the given type.

Parameters **type** (*str*) – `'ide'` or `'scsi'`

Returns `(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` 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 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

Return type int

get_common_subtype(*type*)
Get the sub-type common to all devices of the given type.

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

Returns `None`, if multiple such devices exist and they do not all have the same sub-type.

Returns Subtype string common to all devices of the 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 'fileRef' attribute value of this element

get_id_from_file(*file*)
Get the file ID from the given opaque file object.

Parameters **file** (*xml.etree.ElementTree.Element*) – 'File' element

Returns '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.

Return type dict

Returns `{ profile_name : nic_count }`

get_path_from_file(*file*)
Get the file path from the given opaque file object.

Parameters **file** (*xml.etree.ElementTree.Element*) – 'File' element

Returns 'href' attribute value of this element

get_property_value(*key*)
Get the value of the given property.

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

Returns Value of this property, or `None`

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

Return type dict

Returns `{ profile_name : serial_count }`

get_subtype_from_device(*device*)
Get the sub-type of the given opaque device object.

Parameters **device** – Device object to query

Returns `None`, or string such as 'virtio' or 'lsilogic'

get_type_from_device(*device*)
Get the type of the given device.

Parameters **device** (*OVFItem*) – Device object to query

Returns string such as 'ide' or 'memory'

info_string(*TEXT_WIDTH=79*, *verbosity_option=None*)
Get a descriptive string summarizing the contents of this OVF.

Parameters

- **TEXT_WIDTH** (*int*) – Line length to wrap to where possible.
- **verbosity_option** (*str*) – `'brief'`, `None` (default), or `'verbose'`

Returns Wrapped, appropriately verbose string.

profile_info_list(*TEXT_WIDTH=79*, *verbose=False*)
Get a list describing available configuration profiles.

Parameters

- **TEXT_WIDTH** (*int*) – Line length to wrap to if possible
- **verbose** (*str*) – if True, generate multiple lines per profile

Returns (header, list)

profile_info_string(*TEXT_WIDTH=79*, *verbosity_option=None*)
Get a string summarizing available configuration profiles.

Parameters

- **TEXT_WIDTH** (*int*) – Line length to wrap to if possible
- **verbosity_option** (*str*) – `'brief'`, `None` (default), or `'verbose'`

Returns Appropriately formatted and verbose string.

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 `(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*) – Filename to search from

Returns `(file, disk, ctrl_item, disk_item)`, any or all of which may be `None`

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 `(file, disk, ctrl_item, disk_item)`, any or all of which may be `None`

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 GB 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_subtype (*type*, *profile_list*)
Set the device subtype for the IDE controller(s).

Parameters

- **type** (*str*) – IDE subtype string
- **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).

Since NICs are often named sequentially, this API supports a wildcard option for the final element in `name_list` which can be expanded to automatically assign sequential NIC names. The syntax for the

wildcard option is `{` followed by a number (indicating the starting index for the name) followed by `}`. Examples:

`["eth{0}"]` `Expands to ["eth0", "eth1", "eth2", ...]`

`["mgmt0" "eth{10}"]` `Expands to ["mgmt0", "eth10", "eth11", "eth12", ...]`

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` (*type*, *profile_list*)
Set the hardware type for NICs.

Parameters

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

`set_property_value` (*key*, *value*)
Set the value of the given property (converting value if needed).

Parameters

- **key** (*str*) – Property identifier
- **value** – Value to set for this property

Returns the (converted) value that was set.

`set_scsi_subtype` (*type*, *profile_list*)
Set the device subtype for the SCSI controller(s).

Parameters

- **type** (*str*) – SCSI subtype string
- **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*)
Untar the provided .ova to a the working directory.

Parameters **file** (*str*) – OVA file path

Raises **VMInitError** if the given file does not represent a valid OVA archive.

Returns Path to extracted OVF descriptor

write ()
Write OVF or OVA to `output_file`, if set.

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.

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

networks
The list of network names currently defined in this VM.

Return type list[str]

output_file
OVF or OVA file that will be created or updated by `write()`.

Raises **ValueUnsupportedError** if `detect_type_from_name()` fails

platform
The platform type, as determined from the OVF descriptor.

Type Class object - `GenericPlatform` or a more-specific subclass if recognized as such.

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

For an OVF, this corresponds to the `VirtualSystemType` element.

version_long
Long descriptive version string (XML `FullVersion` element).

version_short
Short descriptive version string (XML `Version` element).

class `OVFNameHelper`(*version*)
Bases: `object`

Helper class for `OVF`.

Provides string constants for easier lookup of various OVF XML elements and attributes.

class `OVFHardware`(*ovf*)
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.

`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 `(instance, ovfitem)`

`find_all_items`(*resource_type=None*, *properties=None*, *profile_list=None*)
Find all items matching the given type, properties, and profiles.

Parameters

- **resource_type** –
- **value] properties** (*dict[property,*) – Property values to match
- **profile_list** (*list*) – List of profiles to filter on

Returns list of `OVFItem` instances

`find_item`(*resource_type=None*, *properties=None*, *profile=None*)
Find the only `OVFItem` of the given `resource_type`.

Parameters

- **resource_type** –
- **properties** –
- **profile** – Single profile ID to search within

Return type `OVFItem` or `None`

Raises LookupError if more than one such Item exists.

`find_unused_instance_id`()
Find the first available `InstanceID` number.

Return type string

`get_item_count`(*resource_type*, *profile*)
Wrapper for `get_item_count_per_profile()`.

Parameters

- **resource_type** (*str*) –
- **profile** (*str*) – Single profile identifier string to look up.

Returns 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*) –
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)

Return type dict[profile, count]

new_item(*resource_type*, *profile_list=None*)
Create a new `OVFItem` of the given type.

Parameters

- **resource_type** (*str*) –
- **profile_list** (*list*) – Profiles the new item should belong to

Returns `(instance, 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*, *property*, *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'
- **property** (*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** – If there are more matching items than entries in `value_list`, set extra items to this value

set_value_for_all_items(*resource_type*, *property*, *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'

- **property** (*str*) – Property name to update
- **new_value** – New value to set the property to
- **profile_list** (*list*) – List of profiles to filter on (default: apply across all profiles)
- **create_new** (*boolean*) – Whether to create a new entry if no items of this `resource_type` presently exist.

update_xml()
Regenerate all Items under the VirtualHardwareSection, if needed.

Will do nothing if no Items have been changed.

class OVFHardwareDataError
Bases: `exceptions.Exception`

The input data used to construct an `OVFHardware` is not sane.

class OVFItem(*ovf*, *item=None*)
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)

add_item(*item*)
Add the given `Item` element to this OVFItem.

Parameters **item** – XML `Item` element

Raises **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`.

generate_items()
Get a list of Item XML elements derived from this object's data.

Return type list[xml.etree.ElementTree.Element]

get(*tag*)
Get the dict associated with the given XML tag, if any.

Parameters **tag** (*str*) – XML tag to look up

Return type dict

Returns Dictionary of values associated with this tag (TODO?)

get_all_values(*tag*)
Get the set of all value strings for the given tag.

Parameters **tag** –

Return type set

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 that the value is associated with
- **profiles** – set of profile names, or None

Returns Value string or `None`

has_profile (*profile*)
Check if this Item exists under the given profile.

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

Return type boolean

remove_profile (*profile*)
Remove all trace of the given profile from this item.

Parameters **profile** – Profile name to remove

set_property (*key*, *value*, *profiles=None*, *overwrite=True*)
Store the value and profiles associated with it for the given key.

Parameters

- **key** (*str*) – Property key
- **value** – Value associated with `key`
- **profiles** (*list[str]*) – If `None`, set for all profiles currently known to this item, else set only for the given list of profiles.
- **overwrite** (*boolean*) – 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.

Raises RuntimeError if validation fails and self-repair is impossible.

ATTRIB_KEY_SUFFIX = ' {Item attribute}'

ELEMENT_KEY_SUFFIX = ' {custom element}'

class **OVFItemDataError**
Bases: `exceptions.Exception`

Data to be added to an `OVFItem` conflicts with existing data.

6.2 Command modules

`COT.submodule`	Parent classes for implementing COT subcommands.
`COT.add_disk`	Module for adding disks to VMs.
`COT.add_file`	Module for adding files to VM definitions.
`COT.deploy`	Module for deploying VM descriptions to a hypervisor to instantiate VMs.

Continued on next page

Table 6.7 – continued from previous page

COT.edit_hardware	Module for editing hardware details of a VM.
COT.edit_product	Module for editing product information in a VM description.
COT.edit_properties	Module for managing VM environment configuration properties.
COT.help	Provide 'help' keyword for COT CLI.
COT.info	Implements "info" subcommand.
COT.inject_config	Implements "inject-config" command.
COT.install_helpers	Implements "install-helpers" command.

6.2.1 COT.submodule module

Parent classes for implementing COT subcommands.

Classes

COTGenericSubmodule	Abstract interface for COT command submodules.
COTReadOnlySubmodule	Class for submodules that do not modify the OVF, such as 'deploy'.
COTSubmodule	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.

create_subparser(*parent*, *storage*)

Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by ArgumentParser.add_subparsers()
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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 (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

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

Returns (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

finished()
Write the current VM state out to disk if requested.

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

Returns (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.

6.2.2 COT.add_disk module

Module for adding disks to VMs.

COTAddDisk(UI)	Add or replace a disk in a virtual machine.
add_disk_worker(vm, UI, DISK_IMAGE[, type, ...])	Worker function for actually adding the disk.

class COTAddDisk(*UI*)

Bases: COT.submodule.COTSubmodule

Add or replace a disk in a virtual machine.

Inherited attributes: UI, package, output

Attributes: disk_image, type, file_id, controller, subtype, address, diskname, description

create_subparser(*parent*, *storage*)

Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by ArgumentParser.add_subparsers()
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

ready_to_run()

Check whether the module is ready to run().

Returns (True, ready_message) or (False, reason_why_not)

run()

Do the actual work of this submodule.

Raises InvalidInputError if ready_to_run() reports False

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.

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
Path to disk image file to add to the VM.

Raises InvalidInputError if the file does not exist.

diskname = None
Name string for the disk.

file_id = None
File identifier to map disk to file.

subtype = None
Controller subtype, such as "virtio".

type = None
Disk type ('harddisk' or 'cdrom').

add_disk_worker(*vm*, *UI*, *DISK_IMAGE*, *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** (OVF or other VMDescription subclass) – The virtual machine being edited.
- **UI** (instance of UI or subclass.) – User interface in effect.
- **DISK_IMAGE** (*str*) – path to disk image to add to the VM.
- **type** (*str*) – Disk 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

6.2.3 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`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

Returns `(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.

6.2.4 COT.deploy module

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

Classes

`COTDeploy`	Semi-abstract class for submodules used to deploy a VM to a hypervisor.
`COTDeployESXi`	Submodule for deploying VMs on ESXi and VMware vCenter/vSphere.

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`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

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(parent, storage)` (to create the main 'deploy' subparser if it doesn't already exist) then call `self.subparsers.add_parser()` to add their own sub-subparser.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

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

configuration
VM configuration profile to use for deployment.

Raise `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.

Raise `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.

subparsers = None
Subparser grouping for hypervisor-specific sub-subparsers.

Subclasses should generally have their `create_subparser()` implementations create their sub-subparsers under `subparsers` and NOT under `parent`.

username = None
Server login username.

vm_name = None
Name of the created virtual machine

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`

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

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

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

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

Returns `(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.

locator = None
vSphere target locator.

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

6.2.5 COT.edit_hardware module

Module for editing hardware details of a VM.

Classes

`COTEditHardware`	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`, `cpus`, `memory`, `nics`, `nic_type`, `mac_addresses_list`, `nic_networks`, `nic_names`, `serial_ports`, `serial_connectivity`, `scsi_subtype`, `ide_subtype`, `virtual_system_type`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

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

run()
Do the actual work of this submodule.

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

cpus
Number of CPUs to set.

ide_subtype = **None**
Subtype string for IDE controllers

mac_addresses_list = **None**
List of MAC addresses to set.

memory
Amount of RAM (in megabytes) to set.

nic_names = **None**
List of NIC name strings

nic_networks = **None**
List of NIC-to-network mappings.

nic_type
NIC type string to set.

nics
Number of NICs to set.

profiles = **None**
Configuration profile(s) to edit.

scsi_subtype = **None**
Subtype string for SCSI controllers

serial_connectivity = **None**
List of serial connection strings.

serial_ports
Serial port count to set.

virtual_system_type = **None**
Virtual system type

6.2.6 COT.edit_product module

Module for editing product information in a VM description.

Classes

COTEditProduct	Edit product information (short version, long version).

class COTEditProduct(*UI*)
Bases: COT.submodule.COTSubmodule

Edit product information (short version, long version).

Inherited attributes: UI, package, output

Attributes: version, full_version

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

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

run()
Do the actual work of this submodule.

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

full_version = None
Long version string.

version = None
Short version string.

6.2.7 COT.edit_properties module

Module for managing VM environment configuration properties.

Classes

`COTEditProperties`	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`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

edit_properties_interactive()
Present an interactive UI for the user to edit properties.

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.

Raise `InvalidInputError` if the file does not exist.

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

6.2.8 `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`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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.

6.2.9 `COT.info` module

Implements "info" subcommand.

class COTInfo(*UI*)
Bases: `COT.submodule.COTGenericSubmodule`

Display VM information string.

Inherited attributes: `UI`

Attributes: `package_list`, `verbosity`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

Returns `(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.

6.2.10 `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`

create_subparser(*parent*, *storage*)
Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

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

Returns `(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
Primary configuration file.

Raises

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

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.

6.2.11 `COT.install_helpers` module

Implements "install-helpers" command.

class `COTInstallHelpers`(*UI*)

Bases: `COT.submodule.COTGenericSubmodule`

Install all helper tools that COT requires.

`create_subparser`(*parent*, *storage*)

Add subparser for the CLI of this submodule.

Parameters

- **parent** (*object*) – Subparser grouping object returned by `ArgumentParser.add_subparsers()`
- **storage** (*dict*) – Dict of { 'label': subparser } to be updated with subparser(s) created, if any.

`install_helper`(*helper*)

Install the given helper module.

Returns (result, message)

`install_manpages`()

Install COT's manual pages.

Returns (result, message)

`run`()

Verify all helper tools and install any that are missing.

6.3 Helper library modules

`COT.data_validation`	Various helpers for data sanity checks.
`COT.platforms`	Handles behavior that varies between guest platforms.

6.3.1 `COT.data_validation` module

Various helpers for data sanity checks.

Exceptions

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

Functions

check_for_conflict	Make sure the list does not contain references to more than one object.
device_address	Parser helper function for device address arguments.
mac_address	Parser helper function for MAC address arguments.
match_or_die	Make sure "first" and "second" are equal or raise an error.
natural_sort	Sort the given list "naturally" rather than in ASCII order.
no_whitespace	Parser helper function for arguments not allowed to contain whitespace.
non_negative_int	Parser helper function for integer arguments that must be 0 or more.
positive_int	Parser helper function for integer arguments that must be 1 or more.
to_string	Get string representation of an object, special-case for XML Element.
validate_int	Parser helper function for validating integer arguments in a range.

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*, *expected*)
Bases: COT.data_validation.ValueUnsupportedError

A numerical input was higher than the highest supported value.

Variables

- **value_type** – descriptive string
- **actual_value** – invalid value that was provided
- **expected_value** – maximum supported value

exception ValueTooLowError(*value_type*, *actual*, *expected*)
Bases: COT.data_validation.ValueUnsupportedError

A numerical input was less than the lowest supported value.

Variables

- **value_type** – descriptive string
- **actual_value** – invalid value that was provided
- **expected_value** – minimum supported value

exception ValueUnsupportedError(*value_type*, *actual*, *expected*)
Bases: COT.data_validation.InvalidInputError

An unsupported value was provided.

Variables

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

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 the object or `None`

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 Validated string (with leading/trailing whitespace stripped)

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** – String to validate

Raises InvalidInputError if string is not a valid MAC address

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

Raises ValueMismatchError if `first != second`

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 Sorted list

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 Validated string (with leading/trailing whitespace stripped)

non_negative_int(*string*)
Parser helper function for integer arguments that must be 0 or more.

Alias for `validate_int()` setting `min` to 0.

positive_int(*string*)
Parser helper function for integer arguments that must be 1 or more.

Alias for `validate_int()` setting `min` to 1.

to_string(*obj*)
Get string representation of an object, special-case for XML Element.

validate_int(*string*, *min=None*, *max=None*, *label='input'*)
Parser helper function for validating integer arguments in a range.

Parameters

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

Returns Validated integer value

Raises

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

6.3.2 `COT.platforms` module

Handles behavior that varies between guest platforms.

Classes

`GenericPlatform`	Generic class for operations that depend on guest platform.
`CSR1000V`	Platform-specific logic for Cisco CSR1000V platform.
`IOSv`	Platform-specific logic for Cisco IOSv.
`IOSXRv`	Platform-specific logic for Cisco IOS XRv platform.
`IOSXRvRP`	Platform-specific logic for Cisco IOS XRv HA-capable RP.
`IOSXRvLC`	Platform-specific logic for Cisco IOS XRv line card.
`NXOSv`	Platform-specific logic for Cisco NX-OSv (Titanium).

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.

classmethod guess_nic_name(*nic_number*)

Guess the name of the Nth NIC for this platform.

Note that this counts from 1, not from 0!

classmethod valid_list_only(*desc*, *val*, *supported_list*)
Error if the given value is not an item in the provided list.

classmethod validate_cpu_count(*cpus*)
Throw an error if the number of CPUs is not a supported value.

classmethod validate_memory_amount(*megabytes*)
Throw an error if the amount of RAM is not supported.

classmethod validate_nic_count(*count*)
Throw an error if the number of NICs is not supported.

classmethod validate_nic_type(*type_string*)
Throw an error if the NIC type string is not supported.

classmethod validate_serial_count(*count*)
Throw an error if the number of serial ports is not supported.

BOOTSTRAP_DISK_TYPE = 'cdrom'

CONFIG_TEXT_FILE = 'config.txt'

LITERAL_CLI_STRING = 'config'

PLATFORM_NAME = '(unrecognized platform, generic)'

SECONDARY_CONFIG_TEXT_FILE = None

class CSR1000V
Bases: COT.platforms.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.

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.

classmethod validate_cpu_count(*cpus*)
CSR1000V supports 1, 2, or 4 CPUs.

classmethod validate_memory_amount(*megabytes*)
Minimum 2.5 GB, max 8 GB.

classmethod validate_nic_count(*count*)
CSR1000V requires 3 NICs and supports up to 26.

classmethod validate_serial_count(*count*)
CSR1000V supports 0-2 serial ports.

CONFIG_TEXT_FILE = 'iosxe_config.txt'

LITERAL_CLI_STRING = 'ios-config'

PLATFORM_NAME = 'Cisco CSR1000V'

class IOSv

Bases: COT.platforms.GenericPlatform

Platform-specific logic for Cisco IOSv.

classmethod guess_nic_name(*nic_number*)
GigabitEthernet0/0, GigabitEthernet0/1, etc.

classmethod validate_cpu_count(*cpus*)
IOSv only supports a single CPU.

classmethod validate_memory_amount(*megabytes*)
IOSv has minimum 192 MB (with minimal feature set), max 3 GB.

classmethod validate_nic_count(*count*)
IOSv supports up to 16 NICs.

classmethod validate_nic_type(*type_string*)
IOSv only supports E1000 NICs.

classmethod validate_serial_count(*count*)
IOSv requires 1-2 serial ports.

BOOTSTRAP_DISK_TYPE = 'harddisk'

CONFIG_TEXT_FILE = 'ios_config.txt'

LITERAL_CLI_STRING = None

PLATFORM_NAME = 'Cisco IOSv'

class IOSXRv

Bases: COT.platforms.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.

classmethod validate_cpu_count(*cpus*)
IOS XRv supports 1-8 CPUs.

classmethod validate_memory_amount(*megabytes*)
Minimum 3 GB, max 8 GB of RAM.

classmethod validate_nic_count(*count*)
IOS XRv requires at least one NIC.

classmethod validate_nic_type(*type_string*)
IOS XRv supports E1000 and virtio NICs.

classmethod validate_serial_count(*count*)
IOS XRv supports 1-4 serial ports.

CONFIG_TEXT_FILE = 'iosxr_config.txt'

LITERAL_CLI_STRING = None

PLATFORM_NAME = 'Cisco IOS XRv'

SECONDARY_CONFIG_TEXT_FILE = 'iosxr_config_admin.txt'

class IOSXRvRP

Bases: COT.platforms.IOSXRv

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

classmethod guess_nic_name(*nic_number*)
Fabric and management only.

- fabric
- MgmtEth0/{SLOT}/CPU0/0

classmethod validate_nic_count(*count*)
Fabric plus an optional management NIC.

PLATFORM_NAME = 'Cisco IOS XRv route processor card'

class IOSXRvLC
Bases: COT.platforms.IOSXRv

Platform-specific logic for Cisco IOS XRv line card.

classmethod guess_nic_name(*nic_number*)
Fabric interface plus slot-appropriate GigabitEthernet interfaces.

- 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.

CONFIG_TEXT_FILE = None

PLATFORM_NAME = 'Cisco IOS XRv line card'

SECONDARY_CONFIG_TEXT_FILE = None

class NXOSv
Bases: COT.platforms.GenericPlatform

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

classmethod guess_nic_name(*nic_number*)
NX-OSv names its NICs a bit interestingly...

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

classmethod validate_cpu_count(*cpus*)
NX-OSv requires 1-8 CPUs.

classmethod validate_memory_amount(*megabytes*)
NX-OSv requires 2-8 GB of RAM.

classmethod validate_nic_type (*type_string*)
NX-OSv supports only E1000 and virtio NICs.

classmethod validate_serial_count (*count*)
NX-OSv requires 1-2 serial ports.

CONFIG_TEXT_FILE = 'nxos_config.txt'

LITERAL_CLI_STRING = None

PLATFORM_NAME = 'Cisco NX-OSv'

6.4 User interface modules

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

6.4.1 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.

choose_from_list (*footer*, *option_list*, *default_value*, *header=''*, *info_list=[]*)
Prompt the user to choose from a list.

Parameters

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

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 String containing 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 Input value

Return type str

get_password(*username*, *host*)
Get password string from the user.

> **Warning:** This stub implementation does not actually interact with the user, but instead always returns `"passwd"`. Subclasses should override this method.

Parameters

- **username** (*str*) – Username the password is associated with
- **host** (*str*) – Host the password is associated with

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.

6.4.2 COT.cli module

CLI entry point for the Common OVF Tool (COT) suite.

Classes

CLI	Command-line user interface for COT.

class CLI (*terminal_width=None*)

Bases: COT.ui_shared.UI

Command-line user interface for COT.

confirm	Prompt user to confirm the requested operation.
create_parser	Create parser object for global cot command.
create_subparsers	Populate the CLI sub-parsers for all known submodules.
fill_examples	Pretty-print a set of usage examples.
fill_usage	Pretty-print a list of usage strings for a COT subcommand.
formatter	Create formatter for log output.
get_input	Prompt the user to enter a string.
get_password	Get password string from the user.
main	Main worker function for COT when invoked from the CLI.
parse_args	Parse the given CLI arguments into a namespace object.
run	Parse the given CLI args then run.
set_verbosity	Enable logging and/or change the logging verbosity level.
terminal_width	The width of the terminal in columns.

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 True (user confirms acceptance) 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.

```
>>> 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
```

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

Returns Examples wrapped appropriately to the `terminal_width()` value. CLI examples will be wrapped with backslashes and a hanging indent.

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.

```
>>> fill_usage('add-file', ["FILE PACKAGE [-o OUTPUT] [-f FILE_ID]"])
  cot add-file --help
  cot add-file FILE PACKAGE [-o OUTPUT]
               [-f FILE_ID]
```

Parameters

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

Returns String containing all usage strings, each appropriately wrapped to the `terminal_width()` value.

formatter(*verbosity=20*)

Create formatter for log output.

We offer different (more verbose) formatting when debugging is enabled, hence this need.

Parameters **verbosity** – Logging level as defined by `logging`.

Returns Formatter object for use with `logging`.

Return type instance of `colorlog.ColoredFormatter`

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 Input value

Return type str

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)

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** – Parser namespace object returned from `parse_args()`.

Return type int

Returns

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 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 Return code from `main()`

set_verbosity(*level*)
Enable logging and/or change the logging verbosity level.

Will call `formatter()` and associate the resulting formatter with logging.

Parameters **level** – Logging level as defined by `logging`

terminal_width
The width of the terminal in columns.

main()
Launch COT from the CLI.

CHAPTER 7

COT.helpers package reference

Provide various non-Python helper programs that COT makes use of.

In general, COT submodules should work through the APIs provided in COT.helpers.api rather than accessing individual helper program classes. This gives us the flexibility to change the specific set of helper programs that are used to provide any given functionality with minimal impact to COT as a whole.

7.1 API

convert_disk_image	Convert the given disk image to the requested format/subformat.
create_disk_image	Create a new disk image at the requested location.
get_checksum	Get the checksum of the given file.
get_disk_capacity	Get the storage capacity of the given disk image.
get_disk_format	Get the disk image format of the given file.

7.2 Exceptions

HelperError	A helper program exited with non-zero return code.
HelperNotFoundError	A helper program cannot be located.

7.3 Helper modules

COT.helpers.api	API for abstract access to third-party helper tools.
COT.helpers.helper	Interface for providers of non-Python helper programs.
COT.helpers.fatdisk	Give COT access to fatdisk for creating and updating FAT32 file systems.
COT.helpers.mkisofs	Give COT access to mkisofs or genisoimage for creating ISO images.
COT.helpers.ovftool	Give COT access to ovftool for validating and deploying OVF to ESXi.
COT.helpers.qemu_img	Give COT access to qemu-img for manipulating disk image formats.
COT.helpers.vmdktool	Give COT access to vmdktool for manipulating compressed VMDK files.

7.3.1 COT.helpers.api module

API for abstract access to third-party helper tools.

Abstracts away operations that require third-party helper programs, especially those that are not available through PyPI.

The actual helper programs are provided by individual classes in this package.

Functions

convert_disk_image	Convert the given disk image to the requested format/subformat.
create_disk_image	Create a new disk image at the requested location.
get_checksum	Get the checksum of the given file.
get_disk_capacity	Get the storage capacity of the given disk image.
get_disk_format	Get the disk image format of the given file.

convert_disk_image (*file_path*, *output_dir*, *new_format*, *new_subformat=None*)
Convert the given disk image to the requested format/subformat.

If the disk is already in this format then it is unchanged; otherwise, will convert to a new disk in the specified output_dir and return its path.

Current supported conversions:

- .vmdk (any format) to .vmdk (streamOptimized)
- .img to .vmdk (streamOptimized)

Parameters

- **file_path** (*str*) – Disk image file to inspect/convert
- **output_dir** (*str*) – Directory to place converted image into, if needed
- **new_format** (*str*) – Desired final format
- **new_subformat** (*str*) – Desired final subformat

Returns

- `file_path`, if no conversion was required
- or a file path in `output_dir` containing the converted image

Raises ValueUnsupportedError if the `new_format` and/or `new_subformat` are not supported conversion targets.

create_disk_image (*file_path*, *file_format=None*, *capacity=None*, *contents=[]*)
Create a new disk image at the requested location.

Either `capacity` or `contents` or both must be specified.

Parameters

- **file_path** (*str*) – Desired location of new disk image
- **file_format** (*str*) – Desired image format (if not specified, this will be derived from the file extension of `file_path`)
- **capacity** – Disk capacity. A string like '16M' or '1G'.
- **contents** (*list*) – List of file paths to package into the created image. If not specified, the image will be left blank and unformatted.

get_checksum (*file_path*, *checksum_type*)
Get the checksum of the given file.

Parameters

- **file_path** (*str*) – Path to file to checksum
- **checksum_type** (*str*) – Supported values are 'md5' and 'sha1'.

Returns String containing hexadecimal file checksum

get_disk_capacity (*file_path*)
Get the storage capacity of the given disk image.

Parameters **file_path** (*str*) – Path to disk image file to inspect

Returns Disk capacity, in bytes

get_disk_format (*file_path*)
Get the disk image format of the given file.

> **Warning:** If `file_path` refers to a file which is not a disk image at all, this function will return `('raw', None)`.

Parameters **file_path** (*str*) – Path to disk image file to inspect.

Returns

`(format, subformat)`

- `format` may be `'vmdk'`, `'raw'`, or `'qcow2'`
- `subformat` may be `None`, or various strings for `'vmdk'` files.

7.3.2 COT.helpers.helper module

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.

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*, *version_args=None*, *version_regexp='([0-9.]+'*)
Bases: `object`

A provider of a non-Python helper program.

Class Properties

PACKAGE_MANAGERS	dict() -> new empty dictionary

Class Methods

apt_install	Try to use `apt-get` to install a package.
port_install	Try to use `port` to install a package.
yum_install	Try to use `yum` to install a package.

Continued on next page

Table 7.6 – continued from previous page

download_and_expand	Context manager for downloading and expanding a .tar.gz file.
find_executable	Wrapper for distutils.spawn.find_executable().

Instance Properties

name	Name of the helper program.
path	Discovered path to the helper.
version	Release version of the associated helper program.

Instance Methods

call_helper	Call the helper program with the given arguments.
install_helper	Install the helper program (abstract method).

__init__ (*name*, *version_args=None*, *version_regexp='([0-9.]+'*)
Initializer.

Parameters

- **name** – Name of helper executable
- **version_args** (*list*) – Args to pass to the helper to get its version. Defaults to ['--version'] if unset.
- **version_regexp** – Regexp to get the version number from the output of the command.

classmethod apt_install (*package*)
Try to use apt-get to install a package.

call_helper (*args*, *capture_output=True*, *require_success=True*)
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.
- **require_success** (*boolean*) – if True, an exception will be raised if the helper exits with a non-zero status code.

Returns Captured stdout/stderr (if capture_output), else None.

classmethod download_and_expand (**args*, ***kwds*)
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 self.download_and_expand("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.

classmethod find_executable(*name*)
Wrapper for `distutils.spawn.find_executable()`.

install_helper()
Install the helper program (abstract method).

Raise `NotImplementedError` as this method must be implemented by a concrete subclass.

classmethod port_install(*package*)
Try to use `port` to install a package.

classmethod yum_install(*package*)
Try to use `yum` to install a package.

PACKAGE_MANAGERS = {'apt-get': '/usr/bin/apt-get', 'yum': None, 'port': None}
Class-level lookup for package manager executables.

name
Name of the helper program.

path
Discovered path to the helper.

version
Release version of the associated helper program.

7.3.3 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`

Helper provider for `fatdisk` (http://github.com/goblinhack/fatdisk).

Methods

`install_helper`	Install `fatdisk`.
`create_raw_image`	Create a new FAT32-formatted raw image at the requested location.

create_raw_image(*file_path*, *contents*, *capacity=None*)
Create a new FAT32-formatted raw image at the requested location.

Parameters

- **file_path** (*str*) – Desired location of new disk image
- **contents** (*list*) – List of file paths to package into the created image.
- **capacity** – (optional) Disk capacity. A string like '16M' or '1G'.

install_helper()
Install `fatdisk`.

7.3.4 COT.helpers.mkisofs module

Give COT access to `mkisofs` or `genisoimage` for creating ISO images.

http://cdrecord.org/

class MkIsoFS

Bases: COT.helpers.helper.Helper

Helper provider for mkisofs and/or genisoimage.

http://cdrecord.org/

Methods

install_helper	Install mkisofs and/or genisoimage.
create_iso	Create a new ISO image at the requested location.

create_iso(*file_path*, *contents*)

Create a new ISO image at the requested location.

Parameters

- **file_path** (*str*) – Desired location of new disk image
- **contents** (*list*) – List of file paths to package into the created image.

install_helper()

Install mkisofs and/or genisoimage.

name

Either mkisofs or genisoimage depending on environment.

path

Find either mkisofs or genisoimage if available.

7.3.5 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/

Methods

install_helper	Install ovftool.
validate_ovf	Use VMware's ovftool program to validate an OVF or OVA.

install_helper()

Install ovftool.

Raise NotImplementedError as VMware does not currently provide any mechanism for automatic download of ovftool.

validate_ovf(*ovf_file*)

Use VMware's ovftool program to validate an OVF or OVA.

This checks the file against the OVF standard and any VMware-specific requirements.

Parameters **ovf_file** (*str*) – File to validate

Returns Output from `ovftool`

Raises

- **HelperNotFoundError** – if `ovftool` is not found.
- **HelperError** – if `ovftool` regards the file as invalid

7.3.6 `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).

Methods

`install_helper`	Install `qemu-img`.
`get_disk_format`	Get the major disk image format of the given file.
`get_disk_capacity`	Get the storage capacity of the given disk image.
`convert_disk_image`	Convert the given disk image to the requested format/subformat.
`create_blank_disk`	Create an unformatted disk image at the requested location.

`convert_disk_image` (*file_path*, *output_dir*, *new_format*, *new_subformat=None*)

Convert the given disk image to the requested format/subformat.

If the disk is already in this format then it is unchanged; otherwise, will convert to a new disk in the specified output_dir and return its path.

Current supported conversions:

•.vmdk (any format) to .vmdk (streamOptimized)

•.img to .vmdk (streamOptimized)

Parameters

- **file_path** (*str*) – Disk image file to inspect/convert
- **output_dir** (*str*) – Directory to place converted image into, if needed
- **new_format** (*str*) – Desired final format
- **new_subformat** (*str*) – Desired final subformat

Returns

- `file_path`, if no conversion was required
- or a file path in `output_dir` containing the converted image

Raises NotImplementedError if the `new_format` and/or `new_subformat` are not supported conversion targets.

`create_blank_disk` (*file_path*, *capacity*, *file_format=None*)

Create an unformatted disk image at the requested location.

Parameters

- **file_path** (*str*) – Desired location of new disk image
- **capacity** – Disk capacity. A string like '16M' or '1G'.
- **file_format** (*str*) – Desired image format (if not specified, this will be derived from the file extension of `file_path`)

get_disk_capacity(*file_path*)
Get the storage capacity of the given disk image.

Parameters **file_path** (*str*) – Path to disk image file to inspect

Returns Disk capacity, in bytes

get_disk_format(*file_path*)
Get the major disk image format of the given file.

> **Warning:** If `file_path` refers to a file which is not a disk image at all, this function will return `'raw'`.

Parameters **file_path** (*str*) – Path to disk image file to inspect.

Returns Disk image format (`'vmdk'`, `'raw'`, `'qcow2'`, etc.)

install_helper()
Install `qemu-img`.

7.3.7 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/

Methods

`install_helper`	Install `vmdktool`.
`convert_disk_image`	Convert the given disk image to the requested format/subformat.

convert_disk_image(*file_path*, *output_dir*, *new_format*, *new_subformat=None*)
Convert the given disk image to the requested format/subformat.

If the disk is already in this format then it is unchanged; otherwise, will convert to a new disk in the specified output_dir and return its path.

Current supported conversions:

•.vmdk (any format) to .vmdk (streamOptimized)

•.img to .vmdk (streamOptimized)

Parameters

- **file_path** (*str*) – Disk image file to inspect/convert
- **output_dir** (*str*) – Directory to place converted image into, if needed

- **new_format** (*str*) – Desired final format
- **new_subformat** (*str*) – Desired final subformat

Returns

- `file_path`, if no conversion was required
- or a file path in `output_dir` containing the converted image

Raises NotImplementedError if the `new_format` and/or `new_subformat` are not supported conversion targets.

`install_helper``()`
Install `vmdktool`.

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*

.

C

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y