

Libvirt 0.7

libvirt-test-API

Virtualization Library

BOOK PUBLISHING TOOL



Publican

Libvirt 0.7 libvirt-test-API

Virtualization Library

Edition 1.4

Copyright © 2010 Red Hat, Inc.

Copyright © 2010 Red Hat, Inc. This material may only be distributed subject to the terms and conditions set forth in the GNU Free Documentation License (GFDL), V1.2 or later (the latest version is presently available at <http://www.gnu.org/licenses/fdl.txt>).

This document explains the components of libvirt-test-API and the workflow used to create test cases and test runs.

1. Understanding libvirt-test-API	1
1.1. About libvirt-test-API.....	1
1.2. Libvirt-test-API framework.....	2
1.3. Libvirt-test-API workflow.....	4
2. Writing a test case	5
2.1. Test case file structure.....	5
2.2. Test case example.....	6
3. Creating a configuration file	11
3.1. Configuration file structure.....	11
3.2. Configuration file examples.....	12
3.2.1. Single test case with arguments that have single values.....	12
3.2.2. Single test case with arguments that have multiple values.....	12
3.2.3. Multiple test cases with arguments that have single values.....	13
3.2.4. Multiple test cases with arguments that have multiple values.....	13
3.2.5. Keyword "only".....	14
3.2.6. Keyword "no".....	14
3.2.7. Keyword "no" without a value.....	15
3.2.8. Keyword "include".....	16
3.2.9. Keyword "times".....	16
3.2.10. Keyword "options" with times.....	17
3.2.11. Keyword "options" with multiprocess.....	17
4. Using libvirt-test-API	19
4.1. Running a test.....	19
4.2. Working with log files.....	19
5. Integrating libvirt-test-API into Autotest	21
A. Revision History	23

Understanding libvirt-test-API

Libvirt-test-API is a powerful test tool designed to complement existing libvirt test tools such as libvirt-TCK. Libvirt-test-API provides a set of simple testing procedures specifically targeting libvirt API functionality. Through the collaboration of existing libvirt test tools together with libvirt-test-API, libvirt testing can now be approached from multiple angles; increasing the extent and comprehensiveness of testing.

The following section provides information about libvirt-test-API and its functionality.

1.1. About libvirt-test-API

Libvirt-test-API is based on libvirt bindings for common languages and it provides the low-level subroutines for libvirt bindings API. These subroutines conform to those specified at www.libvirt.org¹ and can be used in the test cases to create the desired test run.

Libvirt-test-API is:

- **Easy to use:** libvirt-test-API uses a text configuration file to set the test parameters, it then runs the test and collects the logs.
- **Portable:** libvirt-test-API can run locally or be used with another test harness, for example Autotest, without modification.
- **Expandable:** libvirt-test-API provides a repository for the test cases, which can be expanded or further developed by the community.

Libvirt-test-API complements libvirt-TCK by providing the following extended functionality:

Table 1.1. Libvirt-test-API functionality

Functionality	Libvirt-TCK	Libvirt-test-API	Added benefit
Supported programming languages	Perl	Perl, Python, (planned support for Java, Ruby).	Additional choice of programming languages to develop test cases.
Test case execution	Serial	Serial, Parallel.	Option to run test cases in parallel.
Configuration	Can define common configuration parameters for test cases.	Support for common configuration and can define logic for test case assembly.	Generate multiple test cases via a single configuration file.
Test object	Compatibility between libvirt and hypervisor.	Compatibility and functionality of the guest.	Functionality testing of the guest.
Test purpose	Developer focus	End user focus	Additional testing from the end user's perspective.
Code re-usability	To change testing logic, test case needs some code re-written.	Test cases are defined by components. To change testing logic,	Higher code re-usability

¹ <http://www.libvirt.org>

Functionality	Libvirt-TCK	Libvirt-test-API	Added benefit
		only the configuration text file needs to be modified.	

1.2. Libvirt-test-API framework

The diagram below depicts the components of libvirt-test-API and how they connect with one another.

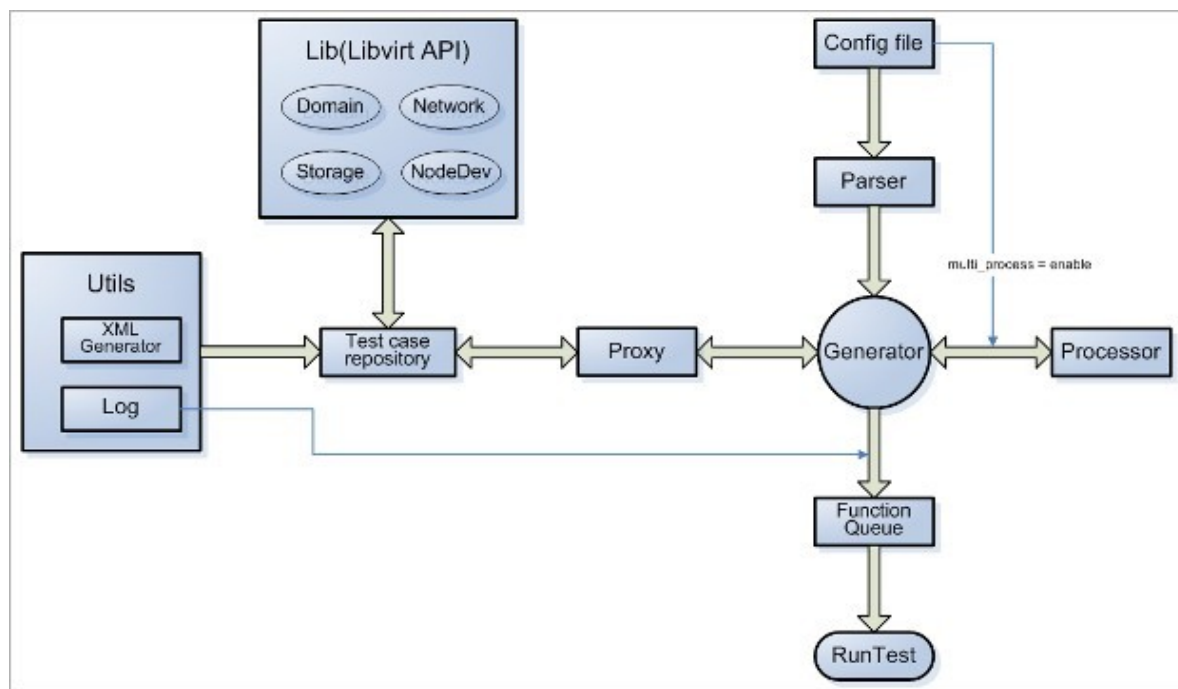


Figure 1.1. Libvirt-test-API framework

The components are described in more detail below.

Configuration file

The configuration file is a '.conf' text file that defines what the test suite does on each test run. The configuration file defines the test cases, their arguments and values. The configuration file structure is as follows:

Example 1.1. Configuration file structure

```

module_1:test_case_1
  argument_1
    value_1
    value_2
  argument_2
    value_1

module_2:test_case_2
  argument_1
    value_2
  
```

The configuration file enables individual test cases to be combined together to achieve the desired test run. To see examples of configuration files, see [Section 3.2, “Configuration file examples”](#).

Parser

The parser converts the configuration file into a list data structure of activities that define the test run.

Activities are a logical structure of test cases, and their sequence is determined by the configuration file.

A test run consists of one or more test activities, and each activity is composed of one or more test cases.

Example 1.2. List data structure

```
[[{'module_1:test_case_1': {'argument_1': 'value_1', 'argument_2':
'value_2'}}, {'module_2:test_case_2': {'argument_1': 'value_1'}}],
[{'module_1:test_case_1': {'argument_1': 'value_2', 'argument_2':
'value_2'}}, {'module_2:test_case_2': {'argument_1': 'value_1'}}]]
```

Generator

The generator is the core of the framework and performs the following functions:

1. Filters the list data structure from the parser and creates a new list of unique test cases. The list of test cases is then sent to the proxy.
2. Separates each activity and removes the module name.
3. Receives a list of callable function objects, created by the proxy from the list in step 1.
4. Combines the lists from step 2 and step 3 to generate a callable subroutine.

The subroutine is now ready to be called by the processor, if multiprocessing is set to enable, or begin a test run immediately from the generator.

Proxy

The proxy examines the list of unique test cases, received from the generator in step 1, and imports each test case from the appropriate module directory.

For example, if the module name is 'domain' then the proxy imports the test case from the **/repos/domain** directory. The proxy then returns a list of the callable function objects of the test cases so the generator can call these functions.

Test case repository

The test case repository is a directory structure that contains a sub-directory for each module. The module directory contains the test cases for that module.

Example 1.3. Test case repository directory structure.

```
repos
|-- module_1
|   |-- test_case_1.py
|-- module_2
|   |-- test_case_2.py
```

Function queue

The function queue contains a list, created by the generator, of the test activities in the form of a callable subroutine.

Processor

The processor handles multiprocessing, if multiprocessing is enabled in the configuration file, then the generator will call the processor to handle the function queue. The processor then generates multiple processes to run the multiple test activities simultaneously. If multiprocessing is disabled, the generator will handle the function queue itself and run the test activities individually.

1.3. Libvirt-test-API workflow

Procedure 1.1. Libvirt-test-API workflow

To use libvirt-test-API:

1. Write a test case.

For information on writing a test case, see [Chapter 2, Writing a test case](#).

2. Create a configuration file.

For information on creating a configuration file, see [Chapter 3, Creating a configuration file](#).

3. Create an environment configuration file

4. Run the test.

For information on running a test, see [Chapter 4, Using libvirt-test-API](#).

Writing a test case

Test cases form the foundation of the test tool. They are the building blocks from which test runs can be created. Test cases are written as Python scripts and define the functions that can be tested. Separating test cases by function allows test cases to be re-used and combined in the configuration file to create the desired test run.

The following section provides details on the test case file structure that must be used in order to be correctly accepted by the test tool.

Note: There are lots of test cases you can use in `libvirttestapi/repos` directory

2.1. Test case file structure

Filename

The name of the test case file must be the same as the name of the main function inside it.

For example, if the main function is `install_guest()`, then test case file must be named `install_guest.py`.

Save the test case to its corresponding directory in `/repos`. For example, if the test case is related to domain then save the file in the `/repos/domain` directory.

Function

The function requires a dictionary (dict) as the argument. The dict uses the function arguments defined in the configuration file.

To write log information to a file, the function also requires a key value pair with the keyname 'logger' and the value is a log object.

Return code

The test case uses the following return codes:

- 0 = success
- 1 = fail
- 2 = skip

Example 2.1. Test case file

```
# install_guest.py
import time
import sys
import os

def install_guest(dict):
    logger = dict['logger']
    print "this is from testcase_repos:domain"
    for eachvars in dict.keys():
        logger.info("the argu is %s" % eachvars)
        time.sleep(1)
        logger.info("the corresponding value is %s" % dict[eachvars])
```

```
logger.info("I am from install_guest log info")
logger.debug("I am from install_guest log debug")
logger.warning("I am from install_guest log warning")
logger.error("I am from install_guest log error")
logger.critical("I am from install_guest log critical")
```

2.2. Test case example

In this example, the test objective is to:

1. Create a new NFS based storage pool and create a volume.
2. Install a new rhel5u4 guest machine on the volume with multiprocess disabled and repeat the test once.

To achieve the test objective, two test cases are required:

1. Initialize the NFS based storage pool, the information of nfsserver is defined in the global.cfg file.

The storage test case is called 'initialize_storage' and is located in **/repos/storage/initialize_storage.py**

2. Install the guest on the volume.

The install test case is called 'domain_install' and is located in **/repos/domain/install_guest.py**

The two test case are independent of each other, which allows the test cases to be re-used and combined with other test cases to create different test runs.

Below is the configuration file, which includes the two test cases.

Example 2.2. Configuration file **case.conf**

```
storage:initialize_storage
    poolname
        $poolname
    pooltype
        netfs
    volname
        $volname
    volformat
        raw
domain:install_guest
    guestname
        $guestname
    guesttype
        $guesttype
    memory
        1048576
    vcpu
        1
options multiprocess=disable times=1
```

Below are the two test case scripts.

Example 2.3. Initialize storage test case /repos/storage/initialize_storage.py

```
#!/usr/bin/env python
import sys
import os
import time
import copy
import shutil
import urllib
import commands

dir = os.path.dirname(sys.modules[__name__].__file__)
absdir = os.path.abspath(dir)
rootdir = os.path.split(os.path.split(absdir)[0])[0]
sys.path.append(rootdir)

import exception
from lib import connectAPI
from lib import storageAPI
from utils import env_parser
from utils import xmlbuilder
envfile = 'env.ini'

def initialize_storage(dict):
    logger = dict['logger']
    dict['hypertype'] = 'xen'
    envpaser = env_parser.Envpaser(envfile)
    dict['sourcename'] = envpaser.get_value('storage', 'sourcename')
    dict['sourcepath'] = envpaser.get_value('storage', 'sourcepath')
    logger.info('prepare create storage pool')
    xmlobj = xmlbuilder.XmlBuilder()
    poolxml = xmlobj.build_pool(dict)
    logger.debug('dump create storage pool xml:\n%s' % poolxml)
    conn = connectAPI.ConnectAPI.open("xen:///")
    stgobj = storageAPI.StorageAPI(conn)
    logger.info('list current storage pool: %s' % stgobj.storage_pool_list())
    logger.info('define pool from xml description')
    stgobj.define_storage_pool(poolxml)
    logger.info('active storage pool')
    stgobj.active_storage_pool(dict['poolname'])
    logger.info('list current storage pool: %s' % stgobj.storage_pool_list())
    volxml = xmlobj.build_volume(dict)
    logger.debug('dump create storage volume xml:\n%s' % volxml)
    logger.info('prepare create storage volume')
    stgobj.create_storage_volume(dict['poolname'], volxml)
    logger.info('list current storage volume: %s' % stgobj.get_volume_list(dict['poolname']))
    return 0
```

Example 2.4. Install guest test case /repos/domain/install_guest.py

```
#!/usr/bin/env python

import sys
import os
import time
import copy
import shutil
import urllib
import commands

envfile = 'global.cfg'

def prepare_cdrom(*args):
    url, ks, gname, logger = args
    ks_name = os.path.basename(ks)
    new_dir = os.path.join('/tmp', gname)
    os.makedirs(new_dir)
    boot_path = os.path.join(url, 'images/boot.iso')
    boot_iso = urllib.urlretrieve(boot_path, 'boot.iso')[0]
    time.sleep(10)
    shutil.move(boot_iso, new_dir)
    ks_file = urllib.urlretrieve(ks, ks_name)[0]
    shutil.move(ks_file, new_dir)
    shutil.copy('utils/ksiso.sh', new_dir)
    src_path = os.getcwd()
    os.chdir(new_dir)
    shell_cmd = 'sh ksiso.sh %s' % ks_file
    (status, text) = commands.getstatusoutput(shell_cmd)
    if status != 0:
        logger.debug(text)
    os.chdir(src_path)

def install_guest(dict):
    logger = dict['logger']
    gname = dict['guestname']
    dict['ifacetype'] = 'bridge'
    dict['bridge'] = 'xenbr0'
    dict['bootcd'] = '/tmp/%s/custom.iso' % gname

    logger.info('get system environment information')
    envpaser = env_parser.Envpaser(envfile)
    url = envpaser.get_value("guest", gname + "src")
    dict['kickstart'] = envpaser.get_value("guest", gname + "ks")
    logger.debug('install source: \n    %s' % url)
    logger.debug('kickstart file: \n    %s' % dict['kickstart'])

    logger.info('prepare installation booting cdrom')
    prepare_cdrom(url, dict['kickstart'], gname, logger)

    xmlobj = xmlbuilder.XmlBuilder()
    guestinstxml = xmlobj.build_domain_install(dict)
    logger.debug('dump installation guest xml:\n%s' % guestinstxml)

    conn = connectAPI.ConnectAPI.open("xen:///")
    domobj = domainAPI.DomainAPI(conn)
    logger.info('define guest from xml description')
    domobj.define_domain(guestinstxml)

    logger.info('start installation guest ...')
    domobj.start_domain(gname)
```

3):

```
kscri = install_common.get_kscri(rhelnewest, guestos, guestarch, "bootiso", logger)
isolink = install_common.get_iso_link(rhelnewest, guestos, guestarch, logger)
```

```
logger.info('prepare installation...')
cache_folder = envparser.get_value("variables", "domain_cache_folder")
```

```
state = domobj.get_domain_state(gname) logger.debug('current guest state: %s' %state)
```

Below is the environment configuration file.

Example 2.5. Environment configuration file `global.cfg`

```
[guest]
guestname = libvirt-test-api
guesttype = xenfv

[storage]
poolname = test_api_pool
pooltype = netfs
```

Below is the log file that is generated. It is stored in `/log`.

Example 2.6. Test case log file

```

_____ initialize_storage _____
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:31) prepare create storage pool
[2009-09-18 15:38:50] 5141 DEBUG    (initialize_storage:34) dump create storage pool xml:
<?xml version="1.0" ?>
<pool type="netfs"><name>nfspool</name><source><host name="10.66.70.85"/><dir path="/ media/
share"/></source><target><path>/var/lib/xen/images</path></target></pool>
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:39) list current storage pool: []
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:40) define pool from xml
description
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:43) active storage pool
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:45) list current storage pool:
['nfspool']
[2009-09-18 15:38:50] 5141 DEBUG    (initialize_storage:48) dump create storage volume
xml:
<?xml version="1.0" ?>
<volume><name>rhel5u4.img</name><capacity unit="G">10</capacity><allocation>0</
allocation><target><path>/var/lib/xen/images</path><format type="raw"/></target></volume>
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:49) prepare create storage volume
[2009-09-18 15:38:50] 5141 INFO      (initialize_storage:51) list current storage volume:
['rhel5u4.img']
----- initialize_storage pass -----

_____ install_guest _____
[2009-09-18 15:38:50] 5141 INFO      (install_guest:55) get system environment information
[2009-09-18 15:38:50] 5141 DEBUG    (install_guest:59) install source:
http://download.redhat.com/pub/rhel/rel-eng/RHEL5.4-Server-latest/tree-x86_64
[2009-09-18 15:38:50] 5141 DEBUG    (install_guest:60) kiskstart file:
http://10.00.00.01/ks-rhel-5.4-x86_64-noxen-smp-minimal.cfg
[2009-09-18 15:38:50] 5141 INFO      (install_guest:62) prepare installation booting cdrom
[2009-09-18 15:39:01] 5141 DEBUG    (install_guest:67) dump installation guest xml:
<?xml version="1.0" ?>
<domain type="xen"><name>rhel5u4</name><memory>1048576</memory><vcpu>1</
vcpu><os><type>hvm</type><loader>/usr/lib/xen/boot/hvmloder</loader><boot dev="cdrom"/
></os><features><acpi/><apic/><pae/></features><clock offset="utc"/><on_poweroff>destroy</
on_poweroff><on_reboot>restart</on_reboot><on_crash>restart</on_crash><devices><emulator>/
usr/lib64/xen/bin/qemu-dm</emulator><disk device="disk" type="file"><driver name="file"/
><source file="/var/lib/xen/images/rhel5u4.img"/><target bus="ide" dev="hda"/></disk><disk
```

```
device="cdrom" type="file"><driver name="file"/><source file="/tmp/rhel5u4/custom.iso"/  
><target bus="ide" dev="hdc"/><readonly/></disk><interface type="bridge"><source  
bridge="xenbr0"/><script path="vif-bridge"/></interface><console/><input bus="ps2"  
type="mouse"/><graphics keymap="en-us" port="-1" type="vnc"/></devices></domain>  
[2009-09-18 15:39:01] 5141 INFO (install_guest:71) define guest from xml description  
[2009-09-18 15:39:01] 5141 INFO (install_guest:74) start installation guest ...  
[2009-09-18 15:39:02] 5141 DEBUG (install_guest:78) current guest state: running
```

This example shows how simple it is to create two test cases to achieve a complex test. The advantage of combining test cases to produce a complex test is that the test cases can be used repeatedly across other tests.

Creating a configuration file

The configuration file is a text file that specifies the test cases, their arguments and values that are to be used in a test run.

3.1. Configuration file structure

The configuration file uses indentation so that the parser can separate test case names, arguments, and values.



Indentation

An indent level is four spaces, do not use tabs to indent.

The configuration file has the following structure.

- **Test case name**

The first line contains the module name and the test case name separated by a colon and is not indented.

```
module:test_case
```

- **Arguments**

Indent Arguments by four spaces.

```
module:test_case
  argument
```

- **Values**

Indent Values by eight spaces.

```
module:test_case
  argument
    value
```

Keywords

Keywords are specified after values and allow you to include, exclude, and repeat individual test cases in a single test run.

Valid keywords are:

- **no**: excludes activities containing the keyword, or all activities if no keyword is specified.
- **only**: includes only the activities containing the keyword.
- **include**: includes activities containing the keyword. It is useful for re-including activities after using the keyword 'no'.

- **times n** : repeats the test case n times.

Options

Options are specified at the end of all the test cases and provide additional flexibility for designing test runs.

- **options times n** : repeats all the test activities n times.
- **options multiprocessing=enable|disable**: when set to enable, the multiprocessing option allows the test run to be processed at the same time as other multiprocessing test runs. If it is not specified it is disabled by default.

Based on the configuration file, the parser generates a data list of dictionary(key, value) pairs.

3.2. Configuration file examples

The following section provides some simple configuration file examples and explains how the configuration file is parsed into the list data structure.

3.2.1. Single test case with arguments that have single values

Example 3.1. Configuration file

```
Domain:install_guest
  guestname
    rhel5u4
  memory
    1024
  vcpu
    1
```

Single values create a single activity.

Example 3.2. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}}]
]
```

3.2.2. Single test case with arguments that have multiple values

Example 3.3. Configuration file

```
Domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2048
  vcpu
    1
```


Multiple values create an activity for each combination of value.

Example 3.4. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'fedora', 'memory': '2048', 'vcpu': '1'}}],
  [{'domain:install_guest': {'guestname': 'rhel8', 'memory': '2048', 'vcpu': '1'}}],
  [{'domain:install_guest': {'guestname': 'rhel9', 'memory': '2048', 'vcpu': '1'}}]
]
```

3.2.3. Multiple test cases with arguments that have single values

Example 3.5. Configuration file

```
domain:install_guest
  guestname
    rhel5u4
  memory
    1024
  vcpu
    1

storage:initialize_storage
  storagename
    rhel5u4
```

Multiple test cases with arguments that have single values create a single activity.

Example 3.6. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'storage:initialize_storage': {'storagename': 'rhel5u4'}}]
]
```

3.2.4. Multiple test cases with arguments that have multiple values

Example 3.7. Configuration file

```
domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2028
  vcpu
    1

storage:initialize_storage
  storagename
    rhel5u4
```

Multiple test cases with arguments that have multiple values create an activity for each combination of value.

Example 3.8. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'fedora', 'memory': '1024', 'vcpu': '1'}},
  {'storage:initialize_storage': {'storagename': 'rhel5u4'}}],
  [{'domain:install_guest': {'guestname': 'rhel8', 'memory': '1024', 'vcpu': '1'}},
  {'storage:initialize_storage': {'storagename': 'rhel9'}}], [{'domain:install_guest':
  {'guestname': 'rhel5u2', 'memory': '1024', 'vcpu': '1'}},
  {'storage:initialize_storage': {'storagename': 'rhel5u4'}}]
]
```

3.2.5. Keyword "only"

Example 3.9. Configuration file

```
domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2048
  vcpu
    1

domain:shutdown_guest
  guestname
    fedora only fedora
    rhel8 only rhel8
    rhel9 only rhel9
```

The keyword "only" creates a test case only for that keyword.

Example 3.10. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u4'}}],
  [{'domain:install_guest': {'guestname': 'rhel5u3', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u3'}}],
  [{'domain:install_guest': {'guestname': 'rhel5u2', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u2'}}]
]
```

3.2.6. Keyword "no"

Example 3.11. Configuration file

```

domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2048
  vcpu
    1
domain:shutdown_guest
  guestname
    fedora no rhel8|rhel9
    rhel8 no fedora|rhel9
    rhel9 no fedora|rhel8

```

The keyword "no" excludes test cases containing the keyword.

```

[
  [{'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '2048', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u4'}}],
  [{'domain:install_guest': {'guestname': 'rhel5u3', 'memory': '2048', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u3'}}],
  [{'domain:install_guest': {'guestname': 'rhel5u2', 'memory': '2048', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u2'}}]
]

```

Example 3.12. List data structure

3.2.7. Keyword "no" without a value

Example 3.13. Configuration file

```

domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2048
  vcpu
    1
domain:shutdown_guest
  guestname
    fedora no
    rhel8 only rhel8
    rhel9 no

```

The keyword "no" without a value means the activity excludes the test case for that value.

Example 3.14. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'fedora', 'memory': '2048', 'vcpu': '1'}}],
  [{'domain:install_guest': {'guestname': 'rhel8', 'memory': '2048', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel8'}}],
  [{'domain:install_guest': {'guestname': 'rhel9', 'memory': '2048', 'vcpu': '1'}}]
]
```

3.2.8. Keyword "include"

Example 3.15. Configuration file

```
domain:install_guest
  guestname
    fedora
    rhel8
    rhel9
  memory
    2048
  vcpu
    1

domain:shutdown_guest
  guestname
    fedora no
    rhel8 only rhel8
    rhel9 no
shutdowndomain
  domainname
    fedora include
    rhel8 only rhel8
    rhel9 include
```

The keyword "include" includes the test case for that value. This is used to re-include a test case after the keyword "no".

Example 3.16. List data structure

```
[
  [{'domain:install_guest': {'guestname': 'fedora', 'memory': '2048', 'vcpu': '1'}},
  {'shutdowndomain': {'domainname': 'fedora'}}],
  [{'domain:install_guest': {'guestname': 'rhel8', 'memory': '2048', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel8'}}, {'shutdowndomain': {'domainname':
  'rhel8'}}],
  [{'domain:install_guest': {'guestname': 'rhel9', 'memory': '2048', 'vcpu': '1'}},
  {'shutdowndomain': {'domainname': 'rhel9'}}]
]
```

3.2.9. Keyword "times"

Example 3.17. Configuration file

```
domain:install_guest times 2
  guestname
    rhel5u4
  memory
    1024
  vcpu
    1
```

The keyword "times n " after the Test Case repeats the test case n more times

Example 3.18. List data structure

```
[
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}}
]
```

3.2.10. Keyword "options" with times

Example 3.19. Configuration file

```
domain:install_guest
  guestname
    rhel5u4
  memory
    1024
  vcpu
    1

domain:shutdown_guest
  guestname
    rhel5u4

options times 2
```

The keyword "options" with **times n** after all test cases repeats all activities n more times.

Example 3.20. List data structure

```
[
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u4'}},
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u4'}},
  {'domain:install_guest': {'guestname': 'rhel5u4', 'memory': '1024', 'vcpu': '1'}},
  {'domain:shutdown_guest': {'guestname': 'rhel5u4'}}
]
```

3.2.11. Keyword "options" with multiprocessing

Example 3.21. Configuration file

```
domain:install_guest
  guestname
    rhel5u4
  memory
    1024
  vcpu
    1

domain:shutdown_guest
  guestname
    rhel5u4

options times 2

domain:shutdown_guest
  guestname
    rhel5u4
options multiprocess=enable|disable
```

The keyword "options" with **multiprocess=enable** or **disable** turns on or off the multiprocessing feature. If this option is not specified it is disabled by default.

Example not shown.

Using libvirt-test-API

After the appropriate test cases and configuration files have been created for the desired test run libvirt-test-API can run the tests and generate the log file.

The following section provides information about running a test and working with log files

4.1. Installing libvirt-test-API

There are three ways to have libvirt-test-API installed:

1.Installing libvirt-test-API on Fedora only takes one command:

```
# dnf install libvirt-test-api
```

2.Installing via Pip

First, install non-python requirements

```
# dnf install virt-install libvirt qemu-img qemu-kvm
```

Then,install libvirt-test-API via pip

```
# pip install libvirt-test-api
```

3.Manually cloning it from gitlab

```
# git clone https://gitlab.com/libvirt/libvirt-test-API.git
```

4.2. Running a test

```
# libvirt-test-api -c case_name
```

Example 4.1. libvirt-test-api command options

```
-c, --casefile      Specify the configuration file to parse, default is 'case.conf'.
-k, --configfile    Specify the environment configuration file
-t, --template:     Print testcase config file template
-f, --logxml:       Specify log file with type xml,defaults to log.xml in current directory
-l, --log-level:    0 or 1 or 2
-d, --delete-log:   Delete log items
-m, --merge:        Merge two log xmlfiles
-r, --return:       Rerun one or more testsuites
```

Libvirt-test-API runs the test and generates the log file.

4.3. Working with log files

Viewing the log file

Libvirt-test-API automatically generates the log file on the first test run. If it is not specified to create another log file, subsequent test runs will append information to the log file.

The log file contains the following information about the test run:

- **Testrunid:** is a series of numbers that denotes the date and time (YYYYMMDDHHMMSS) of each test run. For example, 20090930151144, meaning 2009/09/30 15:11:44.

- **Testid:** lists tests in the log file starting from 001.

Deleting log information

The switch **-d, --delete-log** in the **python main.py** command can be used to delete a test item, a test run, or all test runs in a log file.

For example:

- To delete the test item (testid) in the test run (testrunid):

```
# libvirt-test-api -d log.xml testrunid testid
```


-
- To delete all test runs:

```
# libvirt-test-api -d log.xml all
```
