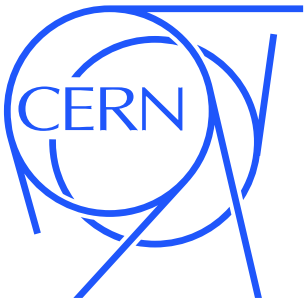

CernVM Release Testing Developer Manual

CernVM Release Testing - Developer Manual

GNU USER

Technical Report
Version: 0.6 (Draft) June 2011

Abstract

The CernVM Release Testing project is a testing infrastructure for CernVM images, the usecase for the project is to provide an automated testing environment, which will install and configure CernVM images, run the set of tests and report the results on a web interface.

Contents

1 Overview

CernVM currently supports images for VirtualBox, VMware, Xen, KVM and Microsoft Hyper-V hypervisors, each new release of a CernVM image needs to be thoroughly tested on each supported platform and hypervisor. The CernVM Release Testing project is designed to meet this requirement by providing an automated testing environment for CernVM images, which will install and configure CernVM images, run the set of tests and report the results on a web interface.

The intent of this document is to provided a reference manual on the CernVM Test Suite Framework for developers, which should provide enough information about the design that developers should easily be able to add new CernVM Release Testing test cases. This developer manual is intended for individuals who have already set up and configured the core components of a Release Testing infrastructure for CernVM image testing, such as the AMD Tapper web server and test clients, including hypervisors.If you already have a CernVM Release Testing infrastructure set up and wish to further expand and develop the code base, then this guide for you.

All the code needed to begin development of the CernVM Test Suite Framework for CernVM image testing is located at the CernVM Release Testing Google Code project page[1] including this document and all other documentation.

While this document is not intended to be a replacement for the AMD Tapper reference manual, the following is a brief description of the Release Testing infrastructure including an introduction to the core component, AMD Tapper [2]. Figure 1.1 consists of a diagram outlining the Tapper Architecture, which consists of test clients and a server, the server is what controls the test clients, gathers results, and then displays the results through a web interface.

The CernVM Test Suite Framework was initially intended to only facilitate the role of "Test Suites", which would execute tests and submit a report file in the form of a "Test Anything Protocol" (TAP) file to the "Test Reports Framework", which is essentially the web server that displays the results of tests. But has since been expanded to comprise the role of the "Test Automation Framework", which deploys, installs, and configures the CernVM images before testing. The most important concept to take away from the diagram is that the CernVM Test Suite Framework includes both the "Test Automation Framework" and "Test Suites", even though it is referred to as a "Test Suite".

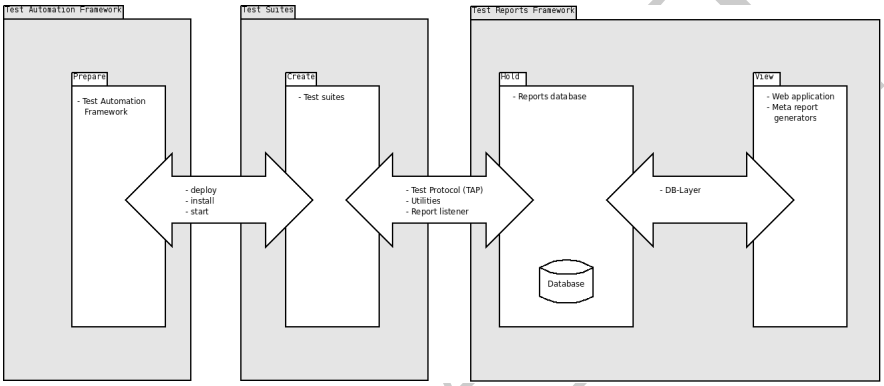

Figure 1.1: Overview of the TAPPER architecture

2 CernVM Test Suite Framework

2.0.1 Framework Design Overview

The CernVM Test Suite Framework was initially designed to facilitate the role of "Test Suites" within the Release Testing infrastructure, which would execute tests and submit a report file in the form of a "Test Anything Protocol" (TAP) file to the "Test Reports Framework". This has since been expanded to compensate for the shortcomings of Tapper and the CernVM Test Suite Framework has since been expanded to comprise the role of the "Test Automation Framework", which deploys, installs, and configures the CernVM images before testing. This is important to understand as the "Precondition Tests" shown in the following diagrams are mostly tests which facilitate the role of the "Test Automation Framework" by ensuring that the CernVM image host environment, and the images themselves are properly configured before executing the actual CernVM Release Testing test cases.

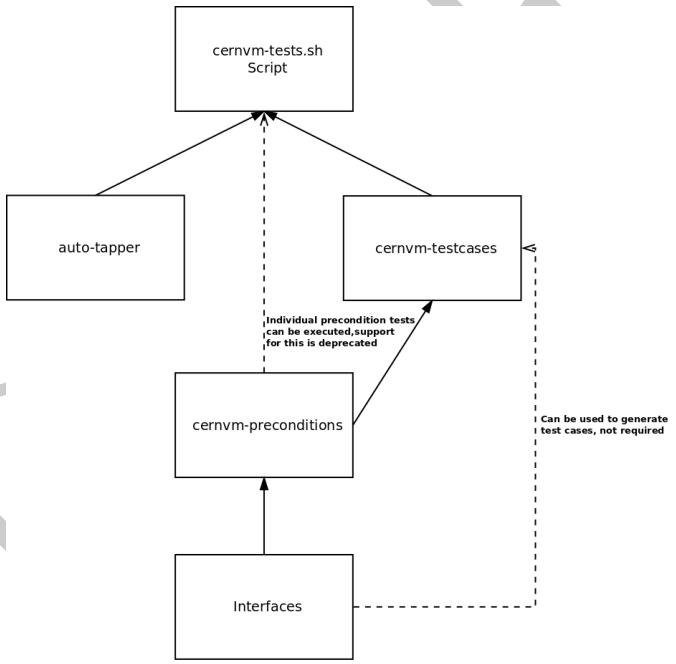

Figure 2.1: Overview of the Proposed CernVM Test Suite Framework

Currently, due to time constraints the optimal CernVM Test Suite Framework design has not been implemented, figure 2.1 is a very simplistic high-level overview of what the *proposed* or intended final architecture is intended to be. The emphasis is on a hierarchical design which is a result in part due to how scope is done in Bash and to limit the functions directly accessed by the **cernvm-tests.sh** script to those provided by auto-tapper and cernvm-testcases. In order for the proposed framework to be implemented, the CernVM test cases must be modular test cases, independent of each other, this has not been implemented yet and as a result the following diagram outlines the current CernVM Test Suite Framework .

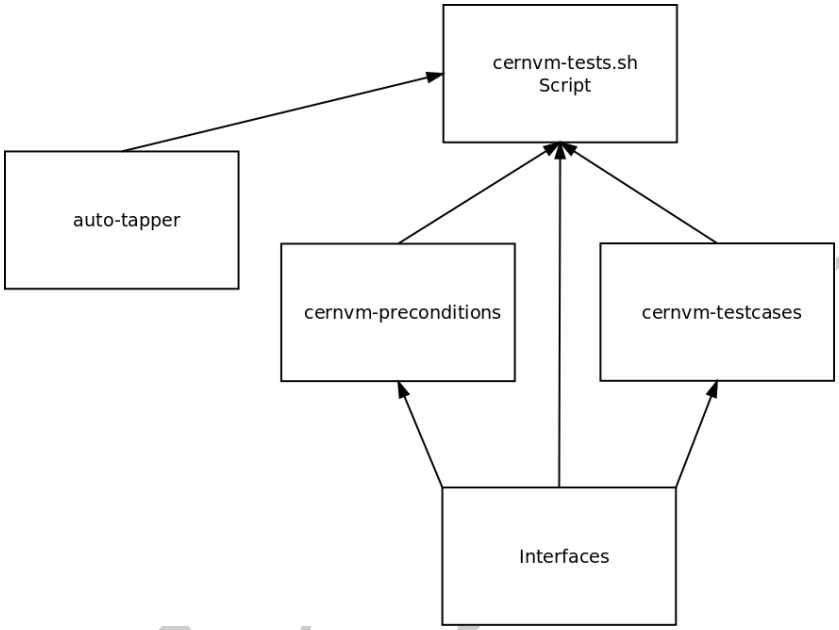

Figure 2.2: Overview of the Current CernVM Test Suite Framework

As shown in figure 2.2 the current architecture of the CernVM Test Suite Framework differs from the proposed framework because the **cernvm-tests.sh script**, *which is the script that executes the set of* CernVM *test cases*, requires both the **cernvm-preconditions** and **cernvm-testcases** files. The **cernvm-preconditions** file is what facilitates the "Test Automation Framework" by ensuring that the host environment and CernVM images are properly configured; the **cernvm-testcases** file is what contains the actual CernVM Release Testing test cases, which are required to test the CernVM image. Inherently, this causes issues as there are precondition tests that must pass before any of the test cases are executed for the results from the test cases to accurate. For example, in order to execute the test case which verifies that the CernVM image has SSH login support, numerous precondition tests

must first be executed which create and configure the CernVM image and verify that it can be started.

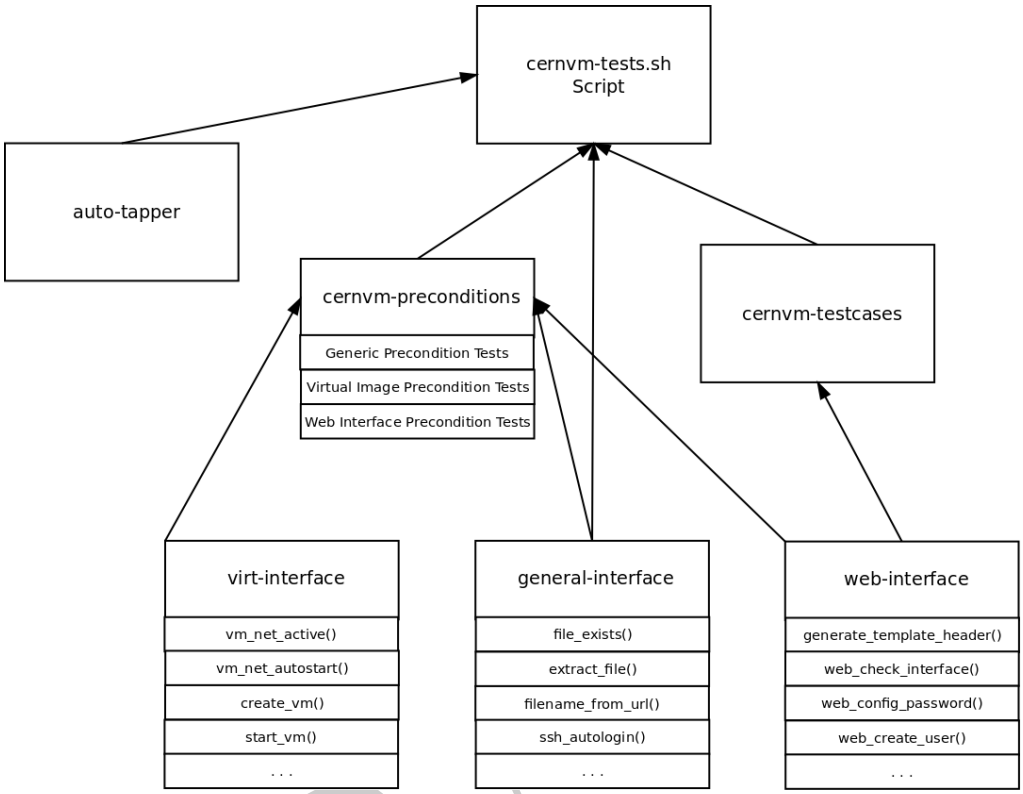

Figure 2.3: Detailed Overview of the Current CernVM Test Suite Framework

The figure 2.3 provides a much more detailed diagram depicting the relations between the different files which are the individual components that make up the current architecture of the CernVM Test Suite Framework . As you can see, the hierarchy still exists to an extent but because of the direct dependency the **cernvm-tests.sh** script has on **cernvm-preconditions** and **cernvm-testcases** there are precondition tests which must be executed first and in the correct order before any of the test cases can be executed.

2.0.2 Precondition Tests

Precondition tests are the tests which fulfil the role of the "Test Automation Framework" referred to in the Tapper architecture figure 1.1 within the CernVM Test Suite Framework . The main purpose of the "Precondition Tests" is to ensure that

the host environment, and the CernVM images themselves are properly configured before executing the actual CernVM Release Testing test cases. The precondition tests configure the host environment and the CernVM images through tests which automate deployment, installation, and configuration of the CernVM images before testing.

Therefore, because the precondition tests are the tests which automate the process of setting up the host environment, the tests must satisfied in order for a CernVM test case to be executed accurately. In a nutshell, they are tests that must be executed, and pass before an actual test case can be executed. Currently, there are three categories of precondition tests,

- Generic Precondition Tests
- Virtual Image Precondition Tests
- Web Interface Precondition Tests

Generic precondition tests, as the name implies, are generic tests which provide functionality to configure the host environment and the CernVM image using methods that are not in the same category as the other two types of precondition tests. For example, a test that downloads and extracts the CernVM image file, would be an example of a generic precondition test. Unlike generic precondition tests, virtual image precondition tests are very specific to configuring the virtualization environment of the CernVM image and involve tests that interact with the libvirt/virsh library through the **virt-interface** such as creating the virtual machine XML definition file and verifying the that the CernVM image can be started. The last category of precondition tests, web interface precondition tests, are unique in that they are tests directly related to configuring the CernVM image through the web interface of the CernVM image. Although the web interface precondition tests could be expanded to include generic "web" tests, currently the precondition test are limited to configuring and controlling the CernVM image through the web interface.

2.0.3 CernVM Test Cases

- They are the tests which are used to make the cernvm test cases modular and independent of each other (ie. can execute boot errors test case before executing the web restart test case)

3 Test Suite Configuration File

The configuration file is essential to setting up the initial CernVM test suite for testing, while most of the default settings provided in the configuration file are sufficient for most CernVM image testing environments, there are still some mandatory settings which **must be configured before testing can begin**. In addition to the mandatory settings that must be specified before tests can be executed, there are also optional configuration settings which provide settings that can override the default settings normally taken when the default configuration file is used, these include options to override the default virtual machine settings specified in the template files.

Each group of settings starts with **CVM**, which is short for CernVM , but then has a unique prefix depending on the category of setting, there are four categories of options for both the mandatory and optional settings that can be provided. The four setting prefixes are **TS, VM, WEB, TC** these denote options that are specific to a category of configuration options. The following is brief summary of each configuration setting prefix, and what category of configurations each prefix applies to.

TS Options which have this prefix are associated with configuration settings specific to the CernVM Release Testing Test Suite

VM Options which have this prefix are associated with configuration settings specific to the CernVM image hypervisor settings, such as the setting for the virtual machine memory

WEB Options which have this prefix are associated with configuration settings specific to the CernVM web interface and configuring the virtual machine through the web interface

TC Options which have this prefix are associated with configuration settings specific to the CernVM Test Cases

3.0.4 Mandatory Settings

In most testing scenarios only the mandatory configuration settings need to be specified such as the hypervisor and the download page, but optional settings are also provided to override internal default settings used by the CernVM Test Suite Framework The following is a list of the mandatory settings that must be configured in order for the tests to work, ensure that you enter valid values, *in lower-case*, for the settings indicated.

- CVM_TS_SUITENAME

  - Must ALWAYS be set, only define once at the top of the configuration file, usually the default suite name given in the test suite configuration file is fine

- CVM_TS_SUITEVERSION
  - Must ALWAYS be set, only define once at the top of the configuration file, reflects the release version number of the test suite framework, the default suite version given in the test suite configuration should only be changed if you make modifications to the test suite framework which differentiate it from the version released on Google Code.

- CVM_TS_REPORT_SERVER
  - Must ALWAYS be set, only define once at the top of the configuration file, this is the ip address or hostname of the Tapper report server which the reports from the test results are sent to

- CVM_TS_DOWNLOAD_PAGE
  - Must ALWAYS be set, normally the default url provided in the configuration file is accurate, but in the event that the internal CernVM image release page is relocated then this url must be changed.

- CVM_VM_HYPERVISOR
  - Must ALWAYS be set, MUST be the first setting before the rest of the mandatory and optional settings specific to the hypervisor are set
  - Valid values (case sensitive) are **kvm,vbox,vmware**

- CVM_VM_TEMPLATE
  - Must ALWAYS be set, normally the default template provided in the configuration file should not be changed, only change this to use a custom template file for the CernVM image
  - The custom template file *must be placed within the templates folder*

- CVM_VM_NET_TEMPLATE
  - Must ALWAYS be set, normally the default network template provided in the configuration file should not be changed, only change this to use a custom network template file for the CernVM image
  - The custom network template file *must be placed within the templates folder*
  - The network template file, **only applies to kvm and virtualbox**

- CVM_VM_IMAGE_VERSION
  - Must ALWAYS be set, MUST be defined for the HYPERVISOR entry in the configuration file, specifies the version of the CernVM image to use from the release page

- CVM_VM_IMAGE_TYPE
  - Must ALWAYS be set, MUST be defined for the HYPERVISOR entry in the configuration file, specifies the type of CernVM image, such as desktop, basic, head node, etc
  - Valid image types supported, (case sensitive) are **basic and desktop**
- CVM_VM_ARCH
  - Must ALWAYS be set, MUST be defined for the HYPERVISOR entry in the configuration file, specifies the architecture of the CernVM image
  - Valid architectures (case sensitive) are **x86 and x86_64**

3.0.5 Optional Settings

In most testing scenarios only the mandatory configuration settings need to be specified such as the hypervisor and the download page, but optional settings are also provided to override internal default settings used by the CernVM Test Suite Framework The following is a list of the optional settings that may be specified to override the default settings, the optional settings must be configured for each of the HYPERVISOR settings defined in the configuration file. The optional settings are separated primarily into four categories, host settings, virtual machine settings, web interface settings, and test case settings.

Again, only the mandatory settings are required to be specified in order for the tests to work, the optional settings can be ignored completely and the test suite scripts should still execute correctly. Therefore, optional settings should only be specified by advanced users as improper optional settings can cause precondition tests to return failures, *it is only recommended that you start configuring optional settings after verifying the results of the scripts using only the mandatory settings.*

Optional Host Settings

- CVM_TS_IMAGES_DIR
  - The root directory for the location of the CernVM images and all configuration files and settings, by default /usr/share/images on Linux/OS X systems and `C:\users\default\application data\images` on Windows systems
- CVM_TS_OSNAME
  - The name of the host operating system, such as Red Hat 5, OS X Snow Leopard, or Windows 7, configure accordingly
  - *Support may be added eventually to automatically configure OSNAME*
- CVM_TS_HOSTNAME
  - The hostname of the system, determined automatically by the script, only set this if you wish to override the default hostname of the system

Optional Virtual Machine Settings

The following are the optional virtual machine settings which can be specified to override the default settings used by the CernVM Test Suite Framework these default virtual machine settings used by the framework are based on the virtual machine XML template definition files defined in the templates directory.

- CVM_VM_IMAGE_RELEASE_ID
  - Overrides the default configuration setting, which uses the most recent release id of a CernVM image, with the specific release id of the CernVM image to download
  - The release id is used to help better identifying the image, as for each new release added that is the same version, the release id will have incremented to indicate that it is a newer release of the same image version
- CVM_VM_NAME
  - Overrides the default name of the virtual machine set by the virtual machine template XML defintion file
  - It is recommended that this setting is specified if testing multiple versions of the same CernVM image, for example a name such as "cernvm-vbox-2.4.0" would help differentiate between other versions
- CVM_VM_CPUS
  - Overrides the default number of cpus, which is one cpu by default, set by the virtual machine template XML defintion file
  - Valid values are from **1 - 4**, but the number specified cannot exceed the actual number of cores/cpus on the host system
- CVM_VM_MEMORY
  - Overrides the default default amount of memory set by the virtual machine template XML defintion file
  - It is recommended that you specify this value if thrashing occurs on the CernVM image when executing tests due to a lack of memory
  - Valid values are in kilobytes and must be based on an amount of memory in kilobytes that is a multiple of a base value of 2. For example, to increase the memory of a system to 1024 MB, set the value as **1048576**, which is the amount of memory in kilobytes
- CVM_VM_VIDEO_MEMORY
  - Overrides the default amount of video memory set by the virtual machine template XML defintion file
  - It is recommended that you specify this value if display errors occurs on the CernVM image before or when executing tests due to a lack of video memory

  - Valid values are in kilobytes and must be based on an amount of video memory in kilobytes that is a multiple of a base value of 2. For example, to increase the video memory of a system to 64 MB, set the value as **65536**, which is the amount of video memory in kilobytes

- CVM_VM_NET_NAME
  - Overrides the default virtual network name set by the virtual machine template XML defintion file
  - This is the one optional setting **you should never configure**, unless you have manually created a different virtual network for the hypervisor

Optional Web Interface Settings

The following are the optional web interface settings which can be specified to override the default settings used by the CernVM Test Suite Framework such as the CernVM image desktop resolution and the primary experiment group.

- CVM_WEB_ADMIN_USERNAME
  - Overrides the default web inteface administration account user name, which is "admin" by default. This optional settings should not have to be modified unless the CernVM web interface defaults change
- CVM_WEB_ADMIN_DEFAULT_PASS
  - Overrides the default web interface administration account password, which is "password" by default. This optional settings should not have to be modified unless the CernVM web interface defaults change
- CVM_WEB_ADMIN_PASS
  - Overrides the web inteface administration account password set by the test suite scripts with a user defined web interface administration password
  - The password specified **must be six characters or longer**
- CVM_WEB_USER_NAME
  - Overrides the default account name "alice" of the new user created by the test suite scripts through the web interface
  - The user name specified should only contain alphabetical characters
- CVM_WEB_USER_PASS
  - Overrides the default password "VM4l1f3" of the new user created by the test suite scripts through the web interface
  - The password specified **must be six characters or longer**

- CVM_WEB_USER_GROUP
  - Overrides the default group "alice" for the new user created by the test suite scripts through the web interface
  - The group specified must be a valid group available through the web interface, such as "alice"
- CVM_WEB_ROOT_PASS
  - Overrides the default password "VM4l1f3" of the root account on the CernVM image set by the test suite scripts through the web interface
  - The password specified **must be six characters or longer**
- CVM_WEB_STARTXONBOOT
  - Overrides the default CernVM desktop setting set by the test suite scripts through the web interface, which configures X to start on boot
  - The valid values, (lower-case) are either "on" to start X on boot, *which is the default*, or "off" to not start X on boot
- CVM_WEB_RESOLUTION
  - Overrides the default CernVM desktop resolution, **1024x768** set by the test suite scripts through the web interface
  - The valid values are valid resolutions up to a **maximum resolution of 1680x1050**
- CVM_WEB_KEYBOARD_LOCALE
  - Overrides the default CernVM desktop keyboard locale, which is "us" by default, set by the test suite scripts through the web interface
  - The valid values are valid locale settings
- CVM_WEB_EXPERIMENT_GROUP
  - Overrides the default CernVM primary experiment group, which is "ALICE" by default, set by the test suite scripts through the web interface
  - The valid values are one of following group names, **the group name specified must be in UPPERCASE**: ALICE, ATLAS, CMS, LHCB, LCD, NA61, HONE, HEPSOFT, BOSS, GEANT4

Optional Test Case Settings

The following are the optional test case settings which can be specified to override the default settings used by the CernVM Test Suite Framework for executing the CernVM Release Testing test cases.

- CVM_TC_USER_NAME
  - Overrides the default account name "bob" of the new user created through the web interface as part of a CernVM Release Testing test case

  - The user name specified should only contain alphabetical characters

- CVM_TC_USER_PASS
  - Overrides the default password "R00tM3" of the new user created through the web interface as part of a CernVM Release Testing test case
  - The password specified **must be six characters or longer**

4 Test Suite API Reference

4.1 test-suite/cernvm-preconditions

[Generics]

NAME

```
cernvm-preconditions
```

DESCRIPTION

```
This script contains each of the CernVM Release Testing precondition
tests, which are required preconditions that must pass for the results
of test cases to be accurate. The precondition tests have a simple
interface to execute each test and each test returns either a success or
failure, (0 or 1)

More complex precondition tests can be created by combining other
precondition tests as prerequisites for a precondition test
```

TODO

```
CLEAN UP THE FOLLOWING PRECONDITON TESTS AND PLACE THEM IN THIS FILE
    Precondition Test 2 - Verify that virtual machine domain has been created
                          from an xml file

    Precondition Test 3 - Verify that virtual machine can be started

    Precondition Test 4 - Verify that virtual machine has been stopped

    Precondition Test 5 - Verify that the virtual has console support

    Precondition Test 6 - Verify that virtual machine has web interface support

    Precondition Test 7 - Verify that it is possible to login on web interface
```

4.1.1 cernvm-preconditions/configure_image_web

[cernvm-preconditions] [Functions]

NAME

```
configure_image_web
```

DESCRIPTION

```
Precondition Test - Setup and configure the initial CernVM image through the
web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The default username to access web interface
$3 - The default password to access web interface
$4 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
configure_image_web 192.168.1.125 admin password config-image.log
```

TODO

```
Implement a function that uses curl to get the updates from the web server
to determine when the system has rebooted
```

4.1.2 cernvm-preconditions/create_def

[cernvm-preconditions] [Functions]

NAME

```
create_def
```

DESCRIPTION

```
Precondition Test - Create an XML definition file for the virtual machine based
on the template XML definition file and settings defined and return the
location of the xml defintion file created
```

ARGUMENTS

```
$1 - The template file to use
$2 - The directory to save the final xml definition file in
```

RETURN VALUE

```
definitionfile - The location of the xml defintion file created
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
create_def vm-template.xml /root
```

4.1.3 cernvm-preconditions/create_net

[cernvm-preconditions] [Functions]

NAME

```
create_net
```

DESCRIPTION

```
Precondition Test - Verify that the virtual machine network has been created
from an xml file
```

ARGUMENTS

```
$1 - The path to the network XML definition file
$2 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
create_net ./network-definition.xml default
```

4.1.4 cernvm-preconditions/create_net_def

[cernvm-preconditions] [Functions]

NAME

```
create_net_def
```

DESCRIPTION

```
Precondition Test - Create an XML definition file for the virtual machine network
based on the template network XML definition file and settings defined and
return the location of the created xml defintion file
```

ARGUMENTS

```
$1 - The network template file to use
$2 - The directory to save the final network xml definition file in
```

RETURN VALUE

```
netdefinitionfile - The location of the network xml defintion file created
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
create_net_def network-template.xml /root
```

4.1.5 cernvm-preconditions/download_extract

[cernvm-preconditions] [Functions]

NAME

```
download_extract
```

DESCRIPTION

```
Precondition Test - Download and extract the CernVM image, returns the location
of the extracted cernvm image file
```

ARGUMENTS

```
$1 - The CernVM image download url
$2 - The directory to place the downloaded image in
$3 - The name of the log file
```

RETURN VALUE

```
imagelocation - The location of the extracted CernVM image file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
IMAGE_LOCATION=$(download_extract http://someurl/file.vmdk.gz /root dl-extract.log)
```

4.1.6 cernvm-preconditions/image_url

[cernvm-preconditions] [Functions]

NAME

```
image_url
```

DESCRIPTION

```
Precondition Test - Verify that the download page exists and that there is a
valid download url for the CernVM image specified, returns the url to
download the image
```

ARGUMENTS

```
$1 - The CernVM download page url
$2 - The image version
$3 - The hypervisor of the image
$4 - The architecture of the image
$5 - The type of image
```

RETURN VALUE

```
imageurl - The url to download the image
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
IMAGE_URL=$(image_url http://downloadpage.com 2.4.0 kvm x86 desktop)
```

4.1.7 cernvm-preconditions/validate_def_settings

[cernvm-preconditions] [Functions]

NAME

```
validate_def_settings
```

DESCRIPTION

```
Precondition Test - Verify that the mandatory configuration settings for the
virtual machine XML definition file have been provided and are valid
```

ARGUMENTS

```
$1 - The virtual machine XML definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
validate_def_settings ./vm-definition.xml
```

4.1.8 cernvm-preconditions/validate_def_xml

[cernvm-preconditions] [Functions]

NAME

```
validate_def_xml
```

DESCRIPTION

```
Precondition Test - Verify that the XML definition file provided is valid
```

ARGUMENTS

```
$1 - The virtual machine XML definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
validate_def_xml ./vm-definition.xml
```

4.1.9 cernvm-preconditions/validate_net_settings

[cernvm-preconditions] [Functions]

NAME

```
validate_net_settings
```

DESCRIPTION

```
Precondition Test - Verify that the mandatory configuration settings for the
network XML definition file have been provided and are valid
```

ARGUMENTS

```
$1 - The network XML definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
validate_net_settings ./network-definition.xml
```

4.1.10 cernvm-preconditions/verify_autologin_ssh

[cernvm-preconditions] [Functions]

NAME

```
verify_autologin_ssh
```

DESCRIPTION

```
Precondition Test - Enable automatic SSH login to the machine for the user
specified using keys instead of passwords, and verify that it is possible
to login automatically
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
$2 - The username to login with
$3 - The password to login with
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_autologin_ssh 192.168.1.125 root password
```

TODO

```
Implement support to only remove the offending key line from known_hosts
instead of deleting the entire file
```

4.1.11 cernvm-preconditions/verify_exists

[cernvm-preconditions] [Functions]

NAME

```
verify_exists
```

DESCRIPTION

```
Precondition Test - Verify that a file/folder exists
```

ARGUMENTS

```
$1 - The location and name of the file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_exists /root/file.tar.gz
```

4.1.12 cernvm-preconditions/verify_hash

[cernvm-preconditions] [Functions]

NAME

```
verify_hash
```

DESCRIPTION

```
Precondition Test - Verify the hash of a file
```

ARGUMENTS

```
$1 - The location and name of the file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_hash /root/file.tar.gz
```

TODO

```
Implement the verify_hash function later as it is not important at the moment
```

4.1.13 cernvm-preconditions/verify_hypervisor

[cernvm-preconditions] [Functions]

NAME

```
verify_hypervisor
```

DESCRIPTION

```
Precondition Test - Verify that the hypervisor for the current virtual machine
tested is accessible, set the hypervisor URI as a global variable
```

ARGUMENTS

```
$1 - The virtual machine XML definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_hypervisor vm-definition.xml
```

4.1.14 cernvm-preconditions/verify_ssh_login

[cernvm-preconditions] [Functions]

NAME

```
verify_ssh_login
```

DESCRIPTION

```
Precondition Test - Verify that user is able to login via ssh
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
$2 - The username to login with
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_ssh_login 192.168.1.125 root
```

4.1.15 cernvm-preconditions/verify_vm_net

[cernvm-preconditions] [Functions]

NAME

```
verify_vm_net
```

DESCRIPTION

```
Precondition Test - Verify that virtual machine NAT network is active and
set to autostart
```

ARGUMENTS

```
$1 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
verify_vm_net default
```

4.2 test-suite/cernvm-testcases

[Generics]

NAME

```
cernvm-testcases
```

DESCRIPTION

```
This script contains each of the CernVM Release Testing
test cases and provides a simple interface to execute each test
and returns either a success or failure, (0 or 1) which can be
used to generate a TAP report.

More complex test cases can be created by combining other test cases
as prerequisites for the test case
```

NOTES

```
Nearly all of the test cases require the root account on the CernVM image as
many of the files and commands can only be accessed by an account with root
privileges
```

TODO

```
MAKE MANY OF THE TEST CASES HAVE OTHER TEST CASES AS
PREREQUISITES AND THEN IF THEY FAIL REPORT THAT THE TEST CASE
FAILED BECAUSE A PREREQUISITE FAILED, AND WHY THAT PREREQUISITE
FAILED. THIS IS MUCH BETTER THAN HAVING A TEST CASE FAIL DUE
TO ANOTHER DEPENDENCY AND MAKES THE TEST CASES ORDER-INDEPENDENT
IE. FOR check_time(), CALL check_ssh() AND VERIFY THAT SSH IS
FIRST POSSIBLE, THIS GIVES MORE EXPLANATION TO FAILURES RATHER
THAN A FAILURE FOR THE NTPD TIME BEING INCORRECT, WHEN IN REALITY
check_time() COULDN'T SSH TO THE MACHINE
*** THIS IS ESSENTIALLY TAPPER'S YAML STRUCTURE ANYWAYS...
```

4.2.1 cernvm-testcases/check_boot_error

[cernvm-testcases] [Functions]

NAME

```
check_boot_error
```

DESCRIPTION

```
CernVM Test Case - Check for error messages at boot
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
$2 - The name of the boot errors log file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
check_boot_error 192.168.1.125 boot-error.log
```

4.2.2 cernvm-testcases/check_ssh

[cernvm-testcases] [Functions]

NAME

```
check_ssh
```

DESCRIPTION

```
CernVM Test Case - Check login via ssh
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
$2 - The username to login with
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
check_ssh 192.168.1.125 root
```

4.2.3 cernvm-testcases/check_time

[cernvm-testcases] [Functions]

NAME

```
check_time
```

DESCRIPTION

```
CernVM Test Case - Check for correct time / running ntpd
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
check_time 192.168.1.125
```

4.2.4 cernvm-testcases/check_web_restart

[cernvm-testcases] [Functions]

NAME

```
check_web_restart
```

DESCRIPTION

```
CernVM Test Case - Restart through the web interface and check that there
                   are no error messages at boot
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the web reboot logfile
$3 - The name of the boot error logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
check_web_restart 192.168.1.125 web-reboot.log boot-error.log
```

4.3 test-suite/general-interface

[Generics]

NAME

```
general-interface
```

DESCRIPTION

```
This script contains general interface functions that interface with
the host system and provide generic functionality such as checking the
host architecture, getting the host operating system, checking if a file
exists, etc.

These functions can be utilized to create precondition tests and test
cases which require generic functionality that is not part of the
virt or web interface functions
```

4.3.1 general-interface/extract_file

[general-interface] [Functions]

NAME

```
extract_file
```

DESCRIPTION

```
Extracts a file based on extension within the directory it is located in
```

ARGUMENTS

```
$1 - The location and name of the file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
extract_file /root/file.tar.gz
```

4.3.2 general-interface/file_exists

[general-interface] [Functions]

NAME

```
file_exists
```

DESCRIPTION

```
Simple function that checks if a file/folder exists
```

ARGUMENTS

```
$1 - The location and name of the file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
file_exists ./template.xml
```

4.3.3 general-interface/filename_from_header

[general-interface] [Functions]

NAME

```
filename_from_header
```

DESCRIPTION

```
Function that returns the name of a file to be downloaded given a url by
looking at the "Location:" specified in HTTP header
```

ARGUMENTS

```
$1 - The download url of the file
```

RETURN VALUE

```
filename - The name of a file to be downloaded
```

EXAMPLE

```
FILE_NAME=$(filename_from_header http://someurl/file.tar.gz)
```

4.3.4 general-interface/filename_from_url

[general-interface] [Functions]

NAME

```
filename_from_url
```

DESCRIPTION

```
Function that returns the name of a file to be downloaded given a url
```

ARGUMENTS

```
$1 - The download url of the file
```

RETURN VALUE

```
filename - The name of a file to be downloaded
```

EXAMPLE

```
FILE_NAME=$(filename_from_url http://someurl/file.tar.gz)
```

4.3.5 general-interface/find_file

[general-interface] [Functions]

NAME

```
find_file
```

DESCRIPTION

```
Function that finds a file and returns the name and path of a file given the
root directory and the extension of the file
```

ARGUMENTS

```
$1 - The root directory to search for the file
$2 - The extension of the file to look for
```

RETURN VALUE

```
filelocation - The name and path of a file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
FILE_LOCATION=$(find_file /usr/share/images vmdk)
```

4.3.6 general-interface/get_hash

[general-interface] [Functions]

NAME

```
get_hash
```

DESCRIPTION

```
Simple function that returns the hash of a file
```

ARGUMENTS

```
$1 - The location and name of the file
$2 - The type of hash, currently supported hashes are:
     crc32, md5, sha, sha1, sha224, sha256, sha384, sha512
```

RETURN VALUE

```
hash - The hash of the file
```

EXAMPLE

```
HASH=$(get_hash /root/file.tar.gz md5)
```

4.3.7 general-interface/get_ip_address

[general-interface] [Functions]

NAME

```
get_ip_address
```

DESCRIPTION

```
Simple function that returns the ip address from the xml network definition file
```

ARGUMENTS

```
$1 - The network XML definition file
```

RETURN VALUE

```
ipaddress - The ip address defined in the xml network definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
IP_ADDRESS=$(get_ip_address ./network-definition.xml)
```

4.3.8 general-interface/get_net_name

[general-interface] [Functions]

NAME

```
get_net_name
```

DESCRIPTION

```
Simple function that returns the network name from xml network definition file
```

ARGUMENTS

```
$1 - The network XML definition file
```

RETURN VALUE

```
networkname - The network name defined in the xml network definition file
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
NET_NAME=$(get_net_name ./network-definition.xml)
```

4.3.9 general-interface/ssh_autologin

[general-interface] [Functions]

NAME

```
ssh_autologin
```

DESCRIPTION

```
A function which configures automatic SSH login using keys instead of passwords
```

ARGUMENTS

```
$1 - The IP address of the machine to login via ssh
$2 - The username to login with
$3 - The password to login with
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
ssh_autologin 192.168.1.125 root password
```

4.4 test-suite/virt-interface

[Generics]

NAME

```
virt-interface
```

DESCRIPTION

```
This script contains several virtualization functions that
interface with libvirsh and return a success or failure,
which can be used to generate a TAP report.

These functions are well suited for precondition tests to ensure
that virtual machines can be created and controlled before
executing any more tests.
```

4.4.1 virt-interface/connect_virsh

[virt-interface] [Functions]

NAME

```
connect_virsh
```

DESCRIPTION

```
Connect to virsh for the current virtual machine hypervisor URI and display URI
```

ARGUMENTS

```
$1 - The URI of the hypervisor
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
connect_virsh qemu:///system
```

4.4.2 virt-interface/create_vm

[virt-interface] [Functions]

NAME

```
create_vm
```

DESCRIPTION

```
Create a virtual machine from an xml file, verify it has been created
```

ARGUMENTS

```
$1 - The path to the virtual machine definition file
$2 - The name of the virtual machine
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
create_vm ./vm-definition.xml virt-machine
```

4.4.3 virt-interface/create_vm_net

[virt-interface] [Functions]

NAME

```
create_vm_net
```

DESCRIPTION

```
Create a virtual machine network from an xml file, verify it has been created
```

ARGUMENTS

```
$1 - The path to the network definition file
$2 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
create_vm_net ./network-definition.xml default
```

4.4.4 virt-interface/destroy_vm

[virt-interface] [Functions]

NAME

```
destroy_vm
```

DESCRIPTION

```
Destroy a virtual machine, verify it has been removed from virsh
```

ARGUMENTS

```
$1 - The name of the virtual machine
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
destroy_vm virt-machine
```

NOTES

```
The files will still exist, the virtual machine is simply no longer
accesible until it is re-created
```

4.4.5 virt-interface/destroy_vm_net

[virt-interface] [Functions]

NAME

```
destroy_vm_net
```

DESCRIPTION

```
Destroy a virtual machine network, verify it has been removed from virsh
```

ARGUMENTS

```
$1 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
destroy_vm_net default
```

NOTES

```
The network definition files will still exist, network is simply no longer
accesible until it is re-created
```

4.4.6 virt-interface/has_console_support

[virt-interface] [Functions]

NAME

```
has_console_support
```

DESCRIPTION

```
Verify that the virtual machine has console support requires that the virtual
machine has been first started
```

ARGUMENTS

```
$1 - The name of the virtual machine
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
has_console_support virt-machine
```

WARNINGS

```
Support for this function has been deprecated and its use is strongly
discouraged as console support is unnecessary and only supported for KVM
```

4.4.7 virt-interface/start_vm

[virt-interface] [Functions]

NAME

```
start_vm
```

DESCRIPTION

```
Start the virtual machine and verify it started
```

ARGUMENTS

```
$1 - The name of the virtual machine
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
start_vm virt-machine
```

4.4.8 virt-interface/stop_vm

[virt-interface] [Functions]

NAME

```
stop_vm
```

DESCRIPTION

```
Stop the virtual machine and verify it has shutdown
```

ARGUMENTS

```
$1 - The name of the virtual machine
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
stop_vm virt-machine
```

4.4.9 virt-interface/vm_net_active

[virt-interface] [Functions]

NAME

```
vm_net_active
```

DESCRIPTION

```
Set the default network as active and verify it is active
```

ARGUMENTS

```
$1 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
vm_net_active default
```

4.4.10 virt-interface/vm_net_autostart

[virt-interface] [Functions]

NAME

```
vm_net_autostart
```

DESCRIPTION

```
Set the default network to autostart and verify that it is set to autostart
```

ARGUMENTS

```
$1 - The virtual machine network name
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
vm_net_autostart default
```

4.5 test-suite/web-interface

[Generics]

NAME

```
web-interface
```

DESCRIPTION

```
This script contains several  functions that provide an interface to
the CernVM virtual machine web interface and return a success or failure,
which can be used to generate a TAP report.

These functions can be utilized to create test cases in cernvm-testcases or
can be executed individually as precondition tests
```

4.5.1 web-interface/generate_header

[web-interface] [Functions]

NAME

```
generate_header
```

DESCRIPTION

```
Generate an http header using the template header and any additional
header values defined
```

EXAMPLE

```
generate_header
```

NOTES

```
This function should only be called within the scope of a web-interface
function after the TEMPLATE_HEADER has been generated and the
ADDITIONAL_HEADER information unique to the function has been set
```

4.5.2 web-interface/generate_template_header

[web-interface] [Functions]

NAME

```
generate_template_header
```

DESCRIPTION

```
Generate a HTTP template header for the current hypervisor which is a basis
to generate headers for different web-interface functions
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
```

RESULT

```
TEMPLATE_HEADER - Exports the generated HTTP template header globally
```

EXAMPLE

```
generate_template_header 192.168.1.125
```

TODO

```
PERHAPS GENERATE DIFFERENT USER-AGENT BASED ON HOST OS
```

4.5.3 web-interface/web_apply_settings

[web-interface] [Functions]

NAME

```
web_apply_settings
```

DESCRIPTION

```
Apply settings configured for the CernVM image using the CernVM web interface,
which then reboots the CernVM image once completed
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_apply_settings 192.168.1.125 apply-settings.log
```

4.5.4 web-interface/web_check_interface

[web-interface] [Functions]

NAME

```
web_check_interface
```

DESCRIPTION

```
Verify that virtual machine has web interface support
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_check_interface 192.168.1.125 check-interface.log
```

4.5.5 web-interface/web_check_login

[web-interface] [Functions]

NAME

```
web_check_login
```

DESCRIPTION

```
Verify that it is possible to login on web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The web interface username, usually admin
$3 - The web interface password, by default password
$4 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_check_login 192.168.1.125 admin password check-login.log
```

4.5.6 web-interface/web_config_desktop

[web-interface] [Functions]

NAME

```
web_config_desktop
```

DESCRIPTION

```
Configure the CernVM image desktop settings using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - Configure startx on boot, accepted values are on or off
$3 - The CernVM image desktop resolution
$4 - The CernVM image keyboard locale
$5 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_config_desktop 192.168.1.125 on 1024x768 us config-desktop.log
```

4.5.7 web-interface/web_config_group

[web-interface] [Functions]

NAME

```
web_config_group
```

DESCRIPTION

```
Configure the CernVM image appliance group settings using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The appliance primary group, all capitals, only one group may be specified
$3 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_config_group 192.168.1.125 ALICE config-group.log
```

TODO

```
ENABLE AN ARRAY / LIST OF APPLIANCE GROUPS
```

4.5.8 web-interface/web_config_password

[web-interface] [Functions]

NAME

```
web_config_password
```

DESCRIPTION

```
Configure the web interface administrator password using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The new web interface administration password
$3 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_config_password 192.168.1.125 newpassword config-password.log
```

4.5.9 web-interface/web_config_proxy

[web-interface] [Functions]

NAME

```
web_config_proxy
```

DESCRIPTION

```
Configure the proxy settings using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_config_proxy 192.168.1.125 config-proxy.log
```

TODO

```
ADD SUPPORT TO ACTUALLY SPECIFY PROXY SETTINGS
```

4.5.10 web-interface/web_create_user

[web-interface] [Functions]

NAME

```
web_create_user
```

DESCRIPTION

```
Create a new user using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the new user to create
$3 - The password for the new user
$4 - The group for the new user, lowercase
$5 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_create_user 192.168.1.125 newuser password alice create-user.log
```

TODO

```
Add documentation to developer-manual which lists available user groups,
as well, perhaps account for invalid user group and return error
```

4.5.11 web-interface/web_restart

[web-interface] [Functions]

NAME

```
web_restart
```

DESCRIPTION

```
Restart through the web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The name of the web reboot logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_restart 192.168.1.125 web-restart.log
```

4.5.12 web-interface/web_root_password

[web-interface] [Functions]

NAME

```
web_root_password
```

DESCRIPTION

```
Set the root password using the CernVM web interface
```

ARGUMENTS

```
$1 - The hostname or ip address for the web interface
$2 - The new root password
$3 - The current web interface administration password
$4 - The name of the logfile
```

RESULT

```
exitstatus - Sets $? as a zero for success, otherwise sets an error code
```

EXAMPLE

```
web_root_password 192.168.1.125 newpass currentpass root-password.log
```

Index

Bibliography

[1] CernVM Release Testing Google Code Project. https://code.google.com/p/cernvm-release-testing/.

[2] Advanced Micro Devices Inc. AMD Tapper. http://developer.amd.com/zones/opensource/amdtapper/pages/default.aspx/, 2011.

Index