

CERNVM RELEASE TESTING DEVELOPER MANUAL

CernVM Release Testing - Developer Manual



GNU USER

Technical Report
Version: 1.2 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	1
2	CernVM Test Suite Framework	3
2.1	Framework Design Overview	3
2.2	Precondition Tests	6
2.3	CERNVM Test Cases	7
3	Test Suite Configuration File	9
3.1	Mandatory Settings	9
3.2	Optional Settings	11
4	Adding CernVM Test Cases	16
4.1	Adding Test Cases Overview	16
4.2	Example, Adding a New Test Case	16
5	Test Suite API Reference	22
5.1	test-suite/cernvm-preconditions	23
5.1.1	cernvm-preconditions/configure_image_web	24
5.1.2	cernvm-preconditions/create_def	25
5.1.3	cernvm-preconditions/create_net	26
5.1.4	cernvm-preconditions/create_net_def	27
5.1.5	cernvm-preconditions/download_extract	28
5.1.6	cernvm-preconditions/image_url	29
5.1.7	cernvm-preconditions/validate_def_settings	30
5.1.8	cernvm-preconditions/validate_def_xml	31
5.1.9	cernvm-preconditions/validate_net_settings	32
5.1.10	cernvm-preconditions/verify_autologin_ssh	33
5.1.11	cernvm-preconditions/verify_exists	34
5.1.12	cernvm-preconditions/verify_hash	35
5.1.13	cernvm-preconditions/verify_hypervisor	36
5.1.14	cernvm-preconditions/verify_ssh_login	37
5.1.15	cernvm-preconditions/verify_virsh_uri	38
5.1.16	cernvm-preconditions/verify_vm_net	39
5.2	test-suite/cernvm-testcases	40
5.2.1	cernvm-testcases/change_user_group	41
5.2.2	cernvm-testcases/check_boot_error	42
5.2.3	cernvm-testcases/check_cvmfs_automount	43

5.2.4	cernvm-testcases/check_cvmfs_cache	44
5.2.5	cernvm-testcases/check_no_network	45
5.2.6	cernvm-testcases/check_ssh	46
5.2.7	cernvm-testcases/check_time	47
5.2.8	cernvm-testcases/check_web_restart	48
5.2.9	cernvm-testcases/migrate_experiment	49
5.3	test-suite/general-interface	50
5.3.1	general-interface/disable_network_if	51
5.3.2	general-interface/enable_network_if	52
5.3.3	general-interface/extract_file	53
5.3.4	general-interface/file_exists	54
5.3.5	general-interface/filename_from_header	55
5.3.6	general-interface/filename_from_url	56
5.3.7	general-interface/find_file	57
5.3.8	general-interface/get_hash	58
5.3.9	general-interface/get_ip_address	59
5.3.10	general-interface/get_mac_address	60
5.3.11	general-interface/get_net_name	61
5.3.12	general-interface/get_os_name	62
5.3.13	general-interface/get_os_type	63
5.3.14	general-interface/ssh_autologin	64
5.3.15	general-interface/ssh_generate_key	65
5.4	test-suite/testsuite-trace	66
5.4.1	testsuite-trace/add_trace_close	67
5.4.2	testsuite-trace/add_trace_output	68
5.4.3	testsuite-trace/add_trace_results	69
5.4.4	testsuite-trace/add_trace_template	70
5.4.5	testsuite-trace/call	71
5.4.6	testsuite-trace/create_trace_log	72
5.4.7	testsuite-trace/error_msg	73
5.4.8	testsuite-trace/generate_trace_template	74
5.4.9	testsuite-trace/generic_msg	75
5.4.10	testsuite-trace/log_trace_output	76
5.4.11	testsuite-trace/success_msg	77
5.5	test-suite/virt-interface	78
5.5.1	virt-interface/connect_virsh	79
5.5.2	virt-interface/create_vm	80
5.5.3	virt-interface/create_vm_net	81
5.5.4	virt-interface/destroy_vm	82
5.5.5	virt-interface/destroy_vm_net	83
5.5.6	virt-interface/has_console_support	84
5.5.7	virt-interface/start_vm	85
5.5.8	virt-interface/stop_vm	86
5.5.9	virt-interface/vm_net_active	87
5.5.10	virt-interface/vm_net_autostart	88

5.6	test-suite/web-interface	89
5.6.1	web-interface/generate_header	90
5.6.2	web-interface/generate_template_header	91
5.6.3	web-interface/web_apply_settings	92
5.6.4	web-interface/web_check_interface	93
5.6.5	web-interface/web_check_login	94
5.6.6	web-interface/web_config_desktop	95
5.6.7	web-interface/web_config_group	96
5.6.8	web-interface/web_config_password	97
5.6.9	web-interface/web_config_proxy	98
5.6.10	web-interface/web_create_user	99
5.6.11	web-interface/web_restart	100
5.6.12	web-interface/web_root_password	101
Bibliography		102
Index		103

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 provide 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 is 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”.

1 Overview

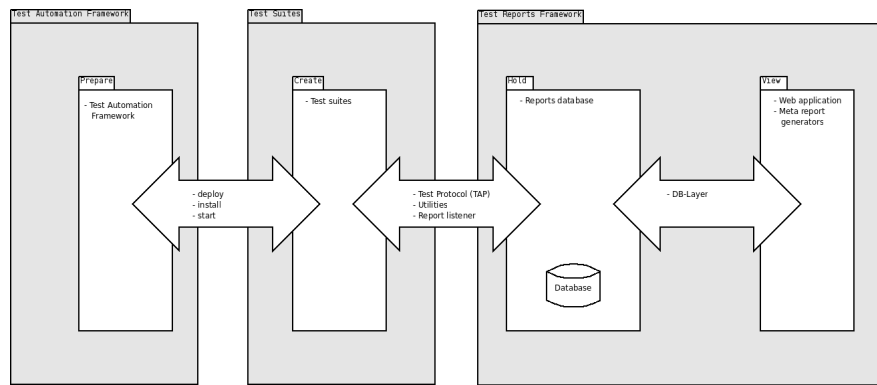


Figure 1.1: Overview of the TAPPER architecture

2 CernVM Test Suite Framework

2.1 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.

2 CERNVM TEST SUITE FRAMEWORK

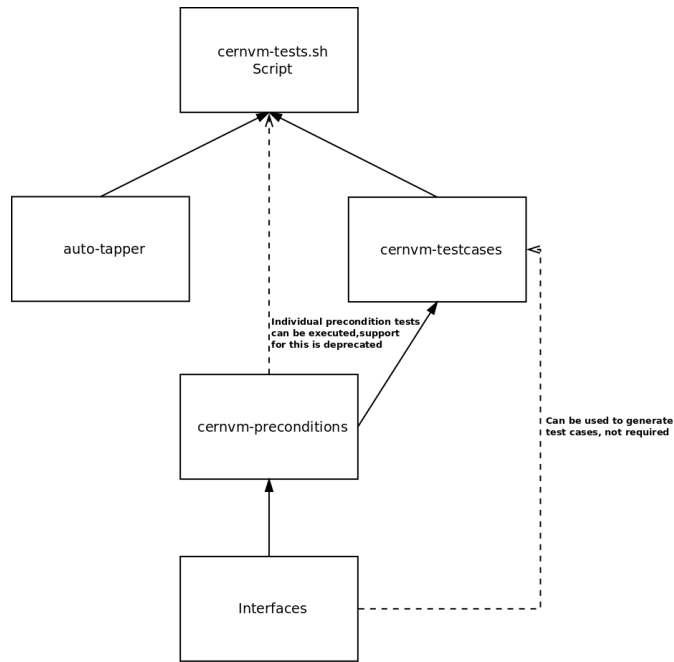


Figure 2.1: Overview of the Proposed CERNVM TEST SUITE FRAMEWORK

2 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 .

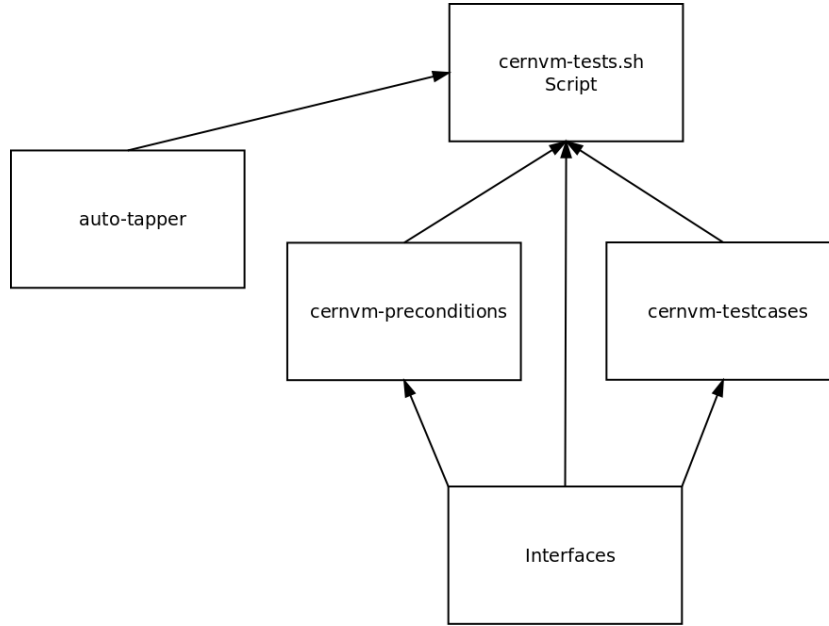


Figure 2.2: Overview of the Current CERNVM TEST SUITE FRAMEWORK

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 be accurate. For example, in order to execute the test case which verifies that the CERNVM image has SSH login support, numerous precondition tests

2 CERNVM TEST SUITE FRAMEWORK

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

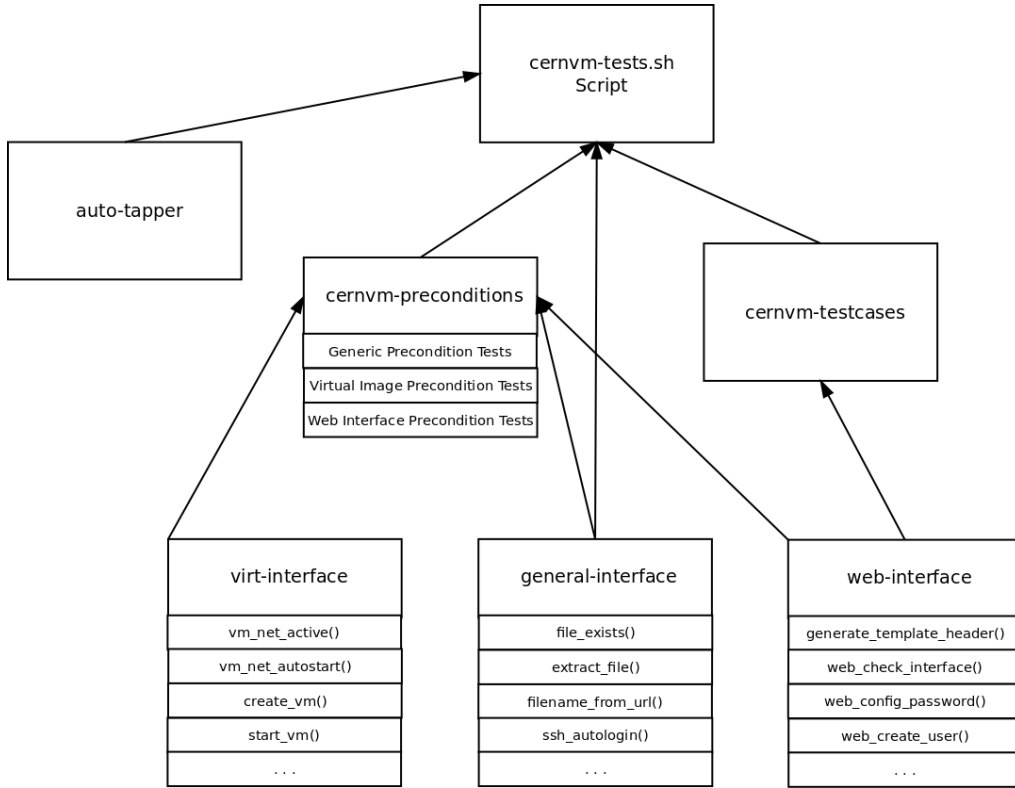


Figure 2.3: Detailed Overview of the Current CERNVM TEST SUITE FRAMEWORK

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.

2.2 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

2 CERNVM TEST SUITE FRAMEWORK

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 be 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 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 is limited to configuring and controlling the CERNVM image through the web interface.

2.3 CernVM Test Cases

The CERNVM Test Cases are simply the tests which fulfil the role of the “CERNVM Test Cases”, these tests are scripted implementations of the CERNVM Test Cases which would have otherwise been executed manually. As referred to in the Current CERNVM TEST SUITE FRAMEWORK figure 2.3. The CERNVM Test Cases are an essential component of the main “cernvm-tests.sh” script, and require both cernvm-preconditions tests as well as interface functions to execute the test cases. The cernvm-precondition functions are essential to executing test cases as the precondition tests are what initially set up and prepare the host environment for executing the actual test cases. In addition, the precondition tests can be used by the CERNVM Test Cases to ensure that the minimum dependencies are met before executing the test case, which allows

2 CERNVM TEST SUITE FRAMEWORK

for modular, or out-of-order testing. This enables a single test to be executed without having any prerequisites for a specific precondition test or test case to have been executed before.

Therefore, the CERNVM Test Cases differ from precondition tests because they are merely tests which automate the otherwise manual process of executing the test cases and checking the CERNVM image. Unlike precondition tests, test cases are not required to pass in order for another CERNVM test case to be executed accurately, any dependencies required for a test case to execute accurately should be instead satisfied by a precondition test. Thus, the results of a test case should not have any influence on the results of another test case, unless they are part of an actual test dependency, any test case requirements such as having network support, or the virtual machine started and configured should be executed by precondition tests.

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

3.1 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

3 Test Suite Configuration File

- 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**, **xen**

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, virtualbox, xen**

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**

3.2 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_OSTYPE

- The type of the host operating system, such as linux, which is automatically determined by the scripts unless specified
- *The valid values, case sensitive, are linux, osx, and windows*

3 Test Suite Configuration File

CVM_TS_OSNAME

- The name and version of the host operating system which is automatically determined by the scripts unless specified, such as "Red Hat 6" or OS X 10.6.8

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 definition 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 definition 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 definition 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

3 Test Suite Configuration File

- 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 definition 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 definition file
- This is the one optional setting **you should never configure**, unless you have manually created a different virtual network for the hypervisor

CVM_VM_MAC_ADDRESS

- Overrides the default MAC address defined in the XML definition file if specified, otherwise the default MAC address set by the virtual machine template XML definition file will be used
- Valid values are valid mac addresses, which are six sets of hexadecimal characters separated by colons, for example 00:21:A9:FE:33:2B

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 interface 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

3 Test Suite Configuration File

CVM_WEB_ADMIN_PASS

- Overrides the web interface 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**

CVM_TC_EXPERIMENT_GROUP

- Overrides the default primary experiment group, “LHCb”, to migrate to as part of a CERNVM RELEASE TESTING test case
- 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

CVM_TC_USER_GROUP

- Overrides the default group “lhcb” to change to for the primary user as part of a CERNVM RELEASE TESTING test case
- The group specified must be a valid group available through the web interface, such as “lhcb”

4 Adding CernVM Test Cases

4.1 Adding Test Cases Overview

The CERNVM Test Cases are simply the tests which fulfil the role of the “CERNVM Test Cases”, these tests are just scripted implementations of the CERNVM Test Cases which would have otherwise been executed manually. The CERNVM Test Cases are an essential component of the main “cernvm-tests.sh” script and require both cernvm-preconditions tests as well as interface functions to execute the test cases. The cernvm-precondition functions are essential to executing test cases as the precondition tests are what initially set up and prepare the host environment for executing the actual test cases. In addition, the precondition tests can be used by the CERNVM Test Cases to ensure that the minimum dependencies are met before executing the test case, which allows for modular, or out-of-order testing. This enables a single test to be executed without having any prerequisites for a specific precondition test or test case to have been executed before.

Therefore, the CERNVM Test Cases differ from precondition tests because they are merely tests which automate the otherwise manual process of executing the test cases and checking the CERNVM image. Unlike precondition tests, test cases are not required to pass in order for another CERNVM test case to be executed accurately, any dependencies required for a test case to execute accurately should be instead satisfied by a precondition test.

Thus, unlike precondition tests, the test cases should only contain the functionality essential to executing the CERNVM Test Cases, anything that could be seen as a prerequisite to executing a test case, such as starting or restarting a virtual machine should not be part of the test case code. *This is important as separating prerequisites from actual test cases enables modular, or out of order testing*; writing prerequisites to test cases as precondition tests allows a single test to be executed independently or the results or execution of another test case.

4.2 Example, Adding a New Test Case

The following will be an example of adding a CERNVM Test Case which verifies that it is possible to login to the CERNVM image using SSH, there will be code samples provided, as well as a detailed explanation of the entire procedure to add a new test case and intergrate it with the “cernvm-tests.sh” script.

1. The first step to take before adding a single line of code is to sit down and analyze any prerequisites for executing the test case, any condition that must be met

4 Adding CERNVM Test Cases

before the actual test case can be executed can be considered a prerequisite and thus would be a precondition test. First start by creating a list of any procedures that would have to be manually executed before actually executing the test case, such as starting the virtual machine and verifying that it has network access.

For the test case used in this example, which verifies that it is possible to login to the CERNVM image using SSH, the list of prerequisites could be listed as something similar to the following.

- Download and extract the CERNVM image
 - Create the virtual machine
 - Configure the virtual machine for testing, such as the amount of memory
 - Start the virtual machine
 - Configure the CERNVM image through the web interface, add a new user, configure experiment group
2. Next, refer to the API reference for a comprehensive list of the precondition tests and interface functions available and look for any precondition tests or functions that satisfy the prerequisites needed to execute the test case. Specifically, refer to the section titled, “test-suite/cernvm-preconditions” [5.1](#), which has every precondition test function documented, including a description of what the precondition test does and the arguments and return values.

For the test case used in this example, all of the prerequisites listed in the previous step can be executed by existing precondition tests and interface functions. Thus, there is a high probability that many of the prerequisites for each test case have been already satisfied by a precondition test. The following is a list of the precondition tests and interface functions that satisfy all of the prerequisites listed in the previous step.

- `download_extract()`
 - `create_def()`
 - `verify_hypervisor()`
 - `create_vm()`
 - `start_vm()`
 - `web_check_interface()`
 - `web_check_login()`
 - `configure_image_web()`
3. Now that the functions necessary to meet all of the prerequisites for the CERNVM test case have been determined, the next step involves determining the variables and values to pass to the precondition tests and other interface functions. After determining the arguments to the functions, determine which configuration options should be specified in the configuration file to be passed as arguments to

4 Adding CERNVM Test Cases

the functions, as almost all of the variables required by the functions should be specified in the configuration file and not set in the `cernvm-tests.sh` script itself. For a complete list of the available configuration options refer to section 3, remember that it is not essential to specify every optional configuration setting as all of the configuration options which are not mandatory have default values.

For the test case used in this is example the KVM CERNVM image will be used to verify SSH access, therefore only the two mandatory configuration options “CVM_TS_REPORT_SERVER” and “CVM_VM_IMAGE_VERSION” would have to be specified in the provided kvm configuration file `content/cernvm-kvm.cfg`. This is because all of the arguments required by the functions listed in the previous step have suitable default values specified in the `cernvm-tests.sh` script if the optional configuration setting is not specified. For example, the following code snippet from `cernvm-tests.sh` demonstrates the suitable default values for the variables required by the `configure_image_web()` function, which configures the CERNVM image through the web interface.

Listing 4.1: Suitable Default Values for the Web Interface

```
##### Optional Web Interface Settings #####
ADMIN_USERNAME="${CVM_WEB_ADMIN_USERNAME:-admin}"
ADMIN_DEFAULT_PASS="${CVM_WEB_ADMIN_DEFAULT_PASS:-password}"
ADMIN_PASS="${CVM_WEB_ADMIN_PASS:-VM411f3}"

# CernVM image user settings , specify the settings for new account
USER_NAME="${CVM_WEB_USER_NAME:-alice}"
USER_PASS="${CVM_WEB_USER_PASS:-VM411f3}"
USER_GROUP="${CVM_WEB_USER_GROUP:-alice}"

# CernVM image root account settings , specify password for root account
ROOT_PASS="${CVM_WEB_ROOT_PASS:-VM411f3}"

# CernVM image desktop settings
STARTXONBOOT="${CVM_WEB_STARTXONBOOT:-on}"
RESOLUTION="${CVM_WEB_RESOLUTION:-1024x768}"
KEYBOARD_LOCALE="${CVM_WEB_KEYBOARD_LOCALE:-us}"

# CernVM image primary group (experiment) settings
EXPERIMENT_GROUP="${CVM_WEB_EXPERIMENT_GROUP:-ALICE}"
```

4. Now that the functions, variables, and configuration options necessary to meet all of the prerequisites for the CERNVM test cases have been determined. The next step involves adding the precondition tests and other interface functions to the main “`cernvm-tests.sh`” script in a form that can be handled by `tapper-autoreport`. While the `cernvm-tests.sh` script is simply a script that has an incremental list of the precondition tests and test cases to be executed, the precondition tests and test cases still need to be integrated with `tapper-autoreport`

4 Adding CERNVM Test Cases

so that the results of the tests can be added to the Test Anything Protocol (TAP) report, which is submitted to the TAPPER Server.

Because many of the test cases share similar prerequisites, such as the virtual machine first being created and started, it is best to place the ordered list of precondition tests within the `cernvm-tests.sh` script before calling the test case function. By placing the calls to precondition tests directly in the `cernvm-tests.sh` script instead of calling them from a test case function, test case dependencies can be avoided as the prerequisites for most of the CERNVM test cases are met before any test case functions are called.

5. After adding the ordered list of precondition tests to the `cernvm-tests.sh` script, the next step is to integrate the precondition tests with `tapper-autoreport` so that the results of the tests can be added to the Test Anything Protocol (TAP) report and submitted to the TAPPER Server. The easiest method of doing this is to catch the exit status of the functions called, which is provided internally by `bash` using the variable `$?` and pass the exit status to the `tapper-autoreport` function `ok`. The `tapper-autoreport` function, “`ok`” takes two arguments, the return code and report message; the most practical method of specifying the report message is to use the precondition test’s function description from the API reference and use an incremental counter for each test.

The following code snippet from `cernvm-tests.sh` is an example of implementing the precondition tests before calling any test case functions and integrating the results of the precondition tests with the auto-tapper reporting facilities. All of the variables used in the function calls are either set based on the options provided in the configuration file or use default values if the optional configuration options are not specified.

Listing 4.2: Adding Precondition Tests and Integrating with Tapper-AutoReport

```
##### Optional CernVM Image Settings #####
IMAGE_RELEASE_ID="${CVM.VM.IMAGE.RELEASE.ID}"
NAME="${CVM.VM.NAME:-cernvm-${HYPERVISOR}-${IMAGE.VERSION}}}"

##### Optional Web Interface Settings #####
ADMIN_USERNAME="${CVM.WEB.ADMIN.USERNAME:-admin}"
ADMIN_DEFAULT_PASS="${CVM.WEB.ADMIN.DEFAULT.PASS:-password}"
ADMIN_PASS="${CVM.WEB.ADMIN.PASS:-VM411f3}"

##### CernVM Image Settings #####
VM.XML.DEFINITION="" # Leave blank, the virtual machine definition file
. . .

# Precondition Test 14 - Verify that virtual machine can be started
start_vm ${VM.XML.DEFINITION} $NAME
```


4 Adding CERNVM Test Cases

```
ok $? "Precondition Test 14 - Verify that virtual machine $VMNAME \
has been started"

. . .

# Precondition Test 17 - Verify that virtual machine has web interface
#                                     support
web_check_interface ${IP_ADDRESS} web_interface.log
ok $? "Precondition Test 17 - Verify that virtual machine $VMNAME has web \
interface support"

# Precondition Test 18 - Verify that it is possible to login on
#                                     web interface
web_check_login ${IP_ADDRESS} $ADMIN_USERNAME $ADMIN_DEFAULT_PASS \
web_interface_login.log
ok $? "Precondition Test 18 - Verify that it is possible to login on \
web interface"

# Precondition Test 19 - Setup and configure the initial CernVM image
#                                     through the web interface
configure_image_web ${IP_ADDRESS} $ADMIN_USERNAME $ADMIN_DEFAULT_PASS \
web_config_image.log
ok $? "Precondition Test 19 - Setup and configure the initial CernVM image \
through the web interface"
```

6. Finally, the last step involves adding the new test case manually, again review the API reference [5.3.8](#) and look for any functions that may already provide the necessary functionality for the test case. In the event that the functionality needed has not already been implemented, create new functions in one of the appropriate interface files. Then, implement a function for the new test case in the “cernvm-testscases” file and call the test case function from “cernvm-tests.sh” and integrate the results with tapper-autoreport.

In some cases, there is a precondition test which already provides the functionality needed to implement the test case and simply needs to be called with arguments specific to the test case. This is the scenario for the test case used in this example, a precondition test already exists to verify that the root account has SSH access, to create the new test case the arguments for the precondition test simply need to be changed.

Listing 4.3: Adding a New Test Case and Integrating with Tapper-AutoReport

```
### To implement the test case call the existing precondition test
```

4 Adding CERNVM Test Cases

```
#### and verify that a specific user, instead of the default root
#### account has SSH access

# Add the test case function to cernvm-tests.sh that verifies SSH access
check_ssh()
{
    verify_ssh_login $1 $2

    return $?
}

#### Finally add the test case to cernvm-tests.sh and
#### Integrate the results of function with taper-autoreport

# CernVM Test Case 1 – Check login via ssh as user created through
# the web interface
check_ssh ${IP_ADDRESS} $USERNAME
ok $? "CernVM_Test_Case_1_-_Check_login_via_ssh_as_user_created_\
through_web_interface"
```

5 Test Suite API Reference

5.1 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

5.1.1 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

5.1.2 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 definition 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 definition file created

RESULT

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

EXAMPLE

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

5.1.3 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`

5.1.4 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 definition 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 definition file created

RESULT

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

EXAMPLE

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


5.1.5 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)
```

5.1.6 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)

5.1.7 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

5.1.8 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

5.1.9 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

5.1.10 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

5.1.11 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

5.1.12 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

5.1.13 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

5.1.14 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
```

5.1.15 cernvm-preconditions/verify_virsh_uri

[cernvm-preconditions] [Functions]

NAME

verify_virsh_uri

DESCRIPTION

Precondition Test - Verify that the URI virsh is connected to matches the URI for the current hypervisor

NOTES

This precondition test is useful for catching a potential libvirt or hypervisor error that is not caught by the scripts or virsh, for example if virsh fails to connect properly to the URI specified, the URI that is returned by virsh will be the default and not match the URI for the current hypervisor being tested

ARGUMENTS

\$1 - The URI of the hypervisor

RESULT

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

EXAMPLE

verify_virsh_uri vmwarews:///session

5.1.16 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

5.2 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...

5.2.1 cernvm-testcases/change_user_group

[cernvm-testcases] [Functions]

NAME

change_user_group

DESCRIPTION

CernVM Test Case - Change the group of the primary user

ARGUMENTS

\$1 - The IP address of the machine to login via ssh
\$2 - The username for the primary user
\$3 - The password for the primary user
\$4 - The new group for the primary user
\$5 - The name of the logfile

RESULT

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

EXAMPLE

change_user_group 192.168.1.125 alice VM411f3 lhcb logfile.log

5.2.2 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

5.2.3 cernvm-testcases/check_cvmfs_automount

[cernvm-testcases] [Functions]

NAME

check_cvmfs_automount

DESCRIPTION

CernVM Test Case - Check that cernvmfs automount scripts works correctly
and is able to mount any experiment group to /cvmfs/

ARGUMENTS

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

\$2 - The appliance primary group, all capitals, only one group may be specified

RESULT

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

EXAMPLE

check_cvmfs_automount 192.168.1.125 ALICE

5.2.4 cernvm-testcases/check_cvmfs_cache

[cernvm-testcases] [Functions]

NAME

check_cvmfs_cache

DESCRIPTION

CernVM Test Case - Check the cvmfs cache list, verify that the cache list is available after restarting the cvmfs daemon

ARGUMENTS

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

\$2 - The name of the logfile

RESULT

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

EXAMPLE

check_cvmfs_cache 192.168.1.125 logfile.log

5.2.5 cernvm-testcases/check_no_network

[cernvm-testcases] [Functions]

NAME

check_no_network

DESCRIPTION

CernVM Test Case - Shutdown the system and disconnect the network, then start the image, it should take longer to boot but the system should not hang on startup

ARGUMENTS

\$1 - The IP address of the machine to login via ssh
\$2 - The name of the virtual machine
\$3 - The virtual machine network name
\$4 - The path to the virtual machine definition file
\$5 - The path to the network definition file

RESULT

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

EXAMPLE

check_no_network 192.168.1.125 vm-name network-name ./vm-def.xml ./network-def.xml

5.2.6 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

5.2.7 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

5.2.8 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

5.2.9 cernvm-testcases/migrate_experiment

[cernvm-testcases] [Functions]

NAME

migrate_experiment

DESCRIPTION

CernVM Test Case - Migrate to another experiment such as LHCb using the web interface and make sure the relative tests are loaded

ARGUMENTS

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

\$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

migrate_experiment 192.168.1.125 ALICE logfile.log

5.3 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

5.3.1 general-interface/disable_network_if

[general-interface] [Functions]

NAME

disable_network_if

DESCRIPTION

Simple function which disables a network interface

ARGUMENTS

\$1 - The name of the network interface

RESULT

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

EXAMPLE

disable_network_if eth0

5.3.2 general-interface/enable_network_if

[*general-interface*] [*Functions*]

NAME

enable_network_if

DESCRIPTION

Simple function which enables a network interface

ARGUMENTS

\$1 - The name of the network interface

RESULT

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

EXAMPLE

enable_network_if eth0

5.3.3 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

5.3.4 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

5.3.5 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)
```

5.3.6 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)

5.3.7 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)
```

5.3.8 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)
```

5.3.9 general-interface/get_ip_address

[*general-interface*] [*Functions*]

NAME

get_ip_address

DESCRIPTION

Simple function that returns the ip address of a virtual machine

ARGUMENTS

\$1 - The current hypervisor for the virtual machine network

\$2 - The MAC address of the virtual machine network connection

\$3 - The network XML definition file, not applicable for VMware

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 kvm 52:54:00:7B:30:75 ./network-definition.xml)
```

```
IP_ADDRESS=$(get_ip_address vmware 00:50:56:C0:00:01)
```


5.3.10 general-interface/get_mac_address

[*general-interface*] [*Functions*]

NAME

get_mac_address

DESCRIPTION

Simple function that returns the mac address from the virtual machine definition file

ARGUMENTS

\$1 - The virtual machine XML definition file

RETURN VALUE

macaddress - The mac address defined in the xml definition file

RESULT

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

EXAMPLE

MAC_ADDRESS=\$(get_mac_address ./kvm-definition.xml)

5.3.11 general-interface/get_net_name

[general-interface] [Functions]

NAME

get_net_name

DESCRIPTION

Simple function that returns the network name for a virtual machine

ARGUMENTS

\$1 - The current hypervisor for the virtual machine network

\$2 - The network XML definition file, not applicable for VMware

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 kvm ./network-definition.xml)
```

```
NET_NAME=$(get_net_name vmware)
```

5.3.12 general-interface/get_os_name

[*general-interface*] [*Functions*]

NAME

get_os_name

DESCRIPTION

Simple function that returns the specific name or version of the OS

ARGUMENTS

\$1 - The type of OS

RETURN VALUE

osname - The name of the OS

EXAMPLE

```
OSNAME=$(get_os_name "linux")
```

5.3.13 general-interface/get_os_type

[*general-interface*] [*Functions*]

NAME

get_os_type

DESCRIPTION

Simple function that returns the type of OS such as linux or osx

RETURN VALUE

ostype - The type of OS, either linux,osx, or windows

EXAMPLE

```
OSTYPE=$(get_os_type)
```

5.3.14 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

5.3.15 general-interface/ssh_generate_key

[*general-interface*] [*Functions*]

NAME

ssh_generate_key

DESCRIPTION

A function which generates an SSH public key

RESULT

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

EXAMPLE

ssh_generate_key

5.4 test-suite/testsuite-trace

[Generics]

NAME

testsuite-trace

DESCRIPTION

This script contains several trace and debugging functions that provide an easy and effective way to log the execution of CernVM test cases, precondition tests, and any other function which enables trace and debugging support.

These functions are well suited for debugging precondition tests and CernVM test cases, to enable debugging of a function simply call the function using "call", to enable more verbose debugging set the trace verbosity level.

5.4.1 testsuite-trace/add_trace_close

[*testsuite-trace*] [*Functions*]

NAME

add_trace_close

DESCRIPTION

Adds the closing marker for the trace of the function call

ARGUMENTS

\$1 - The name of the function that was called

RESULT

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

EXAMPLE

add_trace_close connect_virsh

5.4.2 testsuite-trace/add_trace_output

[*testsuite-trace*] [*Functions*]

NAME

add_trace_output

DESCRIPTION

Adds the data specified to the trace output, which will be logged to file

ARGUMENTS

\$@ - The data to add to the trace output

RESULT

TRACE_OUTPUT - Adds the data specified to TRACE_OUTPUT global variable

INDEX - Sets the global variable index as the next index element of TRACE_OUTPUT

EXAMPLE

```
add_trace_output "${TRACE_TEMPLATE[@]}"
```

5.4.3 testsuite-trace/add_trace_results

[testsuite-trace] [Functions]

NAME

add_trace_results

DESCRIPTION

Adds the results from a traced function call back to the TRACE_OUTPUT heading for the function.

ARGUMENTS

- \$1 - The RETURN VALUE from the function called
- \$2 - The RESULT from the function called
- \$3 - The index of the heading in TRACE_OUTPUT for the function called
- \$4 - The length of the heading for the function called

RESULT

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

EXAMPLE

```
add_trace_results $FUNCTION_RETURN_VALUE $FUNCTION_RESULT 0 12
```

5.4.4 testsuite-trace/add_trace_template

[testsuite-trace] [Functions]

NAME

add_trace_template

DESCRIPTION

Adds the contents of the global TRACE_TEMPLATE to TRACE_OUTPUT, this allows the TRACE_TEMPLATE file to be dynamically re-generated if "call" is used recursively

RESULT

TRACE_OUTPUT - Adds the contents of TRACE_TEMPLATE to the TRACE_OUTPUT global variable
INDEX - Sets the global variable INDEX as the next index element of TRACE_OUTPUT

EXAMPLE

add_trace_template

5.4.5 testsuite-trace/call

[*testsuite-trace*] [*Functions*]

NAME

call

DESCRIPTION

The core of the trace functionality, to register the function with trace and enable debug and logging support prefix the name of the function and its arguments with the trace function "call". This executes the function in the trace debugging environment and logs the results based on the trace verbosity level specified, the trace function "call" is transparent and returns the same exitstatus and return values as the function called.

ARGUMENTS

\$1 - The name of the function to call with trace/debugging support
\$@ - The arguments to the function that is being called

RETURN VALUE

value - The value returned by the function called if applicable

RESULT

exitstatus - Sets the exitstatus as what is returned by the function called, \$? as a zero for success, otherwise sets an error code

EXAMPLE

```
call download_extract http://cern.ch/cernvm-img.tar.gz /usr/share/image logfile.log
```

5.4.6 testsuite-trace/create_trace_log

[*testsuite-trace*] [*Functions*]

NAME

`create_trace_log`

DESCRIPTION

Creates a new trace log file with a title, the date, and basic info

ARGUMENTS

\$1 - The name of the trace log file to create

RESULT

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

EXAMPLE

`create_trace_log cernvm-trace.log`

5.4.7 testsuite-trace/error_msg

[testsuite-trace] [Functions]

NAME

error_msg

DESCRIPTION

Message to report in the trace log file if a function has an error

ARGUMENTS

\$1 - The error message to report

RESULT

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

EXAMPLE

error_msg "The function had an unknown error"

5.4.8 testsuite-trace/generate_trace_template

[testsuite-trace] [Functions]

NAME

generate_trace_template

DESCRIPTION

Generate a template for the trace log file output

ARGUMENTS

\$1 - The name of the function currently being traced

\$@ - The arguments to the function that is being called

RESULT

TRACE_TEMPLATE - Generates a trace template for current function

EXAMPLE

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

5.4.9 testsuite-trace/generic_msg

[testsuite-trace] [Functions]

NAME

generic_msg

DESCRIPTION

A generic message to report in the trace log file

ARGUMENTS

\$1 - A generic message to report

RESULT

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

EXAMPLE

generic_msg "Waiting for the CernVM image to start"

5.4.10 testsuite-trace/log_trace_output

[*testsuite-trace*] [*Functions*]

NAME

log_trace_output

DESCRIPTION

Logs the trace output to the file specified

ARGUMENTS

\$1 - The name of the logfile to store the trace output

RESULT

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

EXAMPLE

log_trace_output \$TRACE_LOGFILE

5.4.11 testsuite-trace/success_msg

[*testsuite-trace*] [*Functions*]

NAME

success_msg

DESCRIPTION

Message to report in the trace log file if a function executes successfully

ARGUMENTS

\$1 - The success message to report

RESULT

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

EXAMPLE

success_msg "The function executed successfully"

5.5 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.

5.5.1 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

RETURN VALUE

URI - The actual URI of the hypervisor that virsh is connected to

RESULT

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

EXAMPLE

connect_virsh qemu:///system

5.5.2 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
```

5.5.3 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
```

5.5.4 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 accessible until it is re-created

5.5.5 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 accessible until it is re-created

5.5.6 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

5.5.7 virt-interface/start_vm

[virt-interface] [Functions]

NAME

start_vm

DESCRIPTION

Start the virtual machine and verify it started

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

start_vm virt-machine

5.5.8 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

5.5.9 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

5.5.10 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

5.6 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

5.6.1 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

5.6.2 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

5.6.3 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

5.6.4 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
```

5.6.5 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

5.6.6 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

5.6.7 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

5.6.8 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

5.6.9 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

5.6.10 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

5.6.11 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

5.6.12 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
```

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

TAPPER Architecture, 1

Functions

- `add_trace_close`, 67
- `add_trace_output`, 68
- `add_trace_results`, 69
- `add_trace_template`, 70
- `call`, 71
- `change_user_group`, 41
- `check_boot_error`, 42
- `check_cvmfs_automount`, 43
- `check_cvmfs_cache`, 44
- `check_no_network`, 45
- `check_ssh`, 46
- `check_time`, 47
- `check_web_restart`, 48
- `configure_image_web`, 24
- `connect_virsh`, 79
- `create_def`, 25
- `create_net`, 26
- `create_net_def`, 27
- `create_trace_log`, 72
- `create_vm`, 80
- `create_vm_net`, 81
- `destroy_vm`, 82
- `destroy_vm_net`, 83
- `disable_network_if`, 51
- `download_extract`, 28
- `enable_network_if`, 52
- `error_msg`, 73
- `extract_file`, 53
- `file_exists`, 54
- `filename_from_header`, 55
- `filename_from_url`, 56
- `find_file`, 57
- `generate_header`, 90
- `generate_template_header`, 91
- `generate_trace_template`, 74
- `generic_msg`, 75
- `get_hash`, 58
- `get_ip_address`, 59
- `get_mac_address`, 60
- `get_net_name`, 61
- `get_os_name`, 62
- `get_os_type`, 63
- `has_console_support`, 84
- `image_url`, 29
- `log_trace_output`, 76
- `migrate_experiment`, 49
- `ssh_autologin`, 64
- `ssh_generate_key`, 65
- `start_vm`, 85
- `stop_vm`, 86
- `success_msg`, 77
- `validate_def_settings`, 30
- `validate_def_xml`, 31
- `validate_net_settings`, 32
- `verify_autologin_ssh`, 33
- `verify_exists`, 34
- `verify_hash`, 35
- `verify_hypervisor`, 36
- `verify_ssh_login`, 37
- `verify_virsh_uri`, 38
- `verify_vm_net`, 39
- `vm_net_active`, 87
- `vm_net_autostart`, 88
- `web_apply_settings`, 92
- `web_check_interface`, 93
- `web_check_login`, 94
- `web_config_desktop`, 95
- `web_config_group`, 96
- `web_config_password`, 97
- `web_config_proxy`, 98
- `web_create_user`, 99

- web_restart, [100](#)
- web_root_password, [101](#)
- Generics
 - cernvm-preconditions, [23](#)
 - cernvm-testcases, [40](#)
 - general-interface, [50](#)
 - testsuite-trace, [66](#)
 - virt-interface, [78](#)
 - web-interface, [89](#)
- unsorted
 - add_trace_close, [67](#)
 - add_trace_output, [68](#)
 - add_trace_results, [69](#)
 - add_trace_template, [70](#)
 - call, [71](#)
 - cernvm-preconditions, [23](#)
 - cernvm-testcases, [40](#)
 - change_user_group, [41](#)
 - check_boot_error, [42](#)
 - check_cvmfs_automount, [43](#)
 - check_cvmfs_cache, [44](#)
 - check_no_network, [45](#)
 - check_ssh, [46](#)
 - check_time, [47](#)
 - check_web_restart, [48](#)
 - configure_image_web, [24](#)
 - connect_virsh, [79](#)
 - create_def, [25](#)
 - create_net, [26](#)
 - create_net_def, [27](#)
 - create_trace_log, [72](#)
 - create_vm, [80](#)
 - create_vm_net, [81](#)
 - destroy_vm, [82](#)
 - destroy_vm_net, [83](#)
 - disable_network_if, [51](#)
 - download_extract, [28](#)
 - enable_network_if, [52](#)
 - error_msg, [73](#)
 - extract_file, [53](#)
 - file_exists, [54](#)
 - filename_from_header, [55](#)
 - filename_from_url, [56](#)
 - find_file, [57](#)
 - general-interface, [50](#)
 - generate_header, [90](#)
 - generate_template_header, [91](#)
 - generate_trace_template, [74](#)
 - generic_msg, [75](#)
 - get_hash, [58](#)
 - get_ip_address, [59](#)
 - get_mac_address, [60](#)
 - get_net_name, [61](#)
 - get_os_name, [62](#)
 - get_os_type, [63](#)
 - has_console_support, [84](#)
 - image_url, [29](#)
 - log_trace_output, [76](#)
 - migrate_experiment, [49](#)
 - ssh_autologin, [64](#)
 - ssh_generate_key, [65](#)
 - start_vm, [85](#)
 - stop_vm, [86](#)
 - success_msg, [77](#)
 - testsuite-trace, [66](#)
 - validate_def_settings, [30](#)
 - validate_def_xml, [31](#)
 - validate_net_settings, [32](#)
 - verify_autologin_ssh, [33](#)
 - verify_exists, [34](#)
 - verify_hash, [35](#)
 - verify_hypervisor, [36](#)
 - verify_ssh_login, [37](#)
 - verify_virsh_uri, [38](#)
 - verify_vm_net, [39](#)
 - virt-interface, [78](#)
 - vm_net_active, [87](#)
 - vm_net_autostart, [88](#)
 - web-interface, [89](#)
 - web_apply_settings, [92](#)
 - web_check_interface, [93](#)
 - web_check_login, [94](#)
 - web_config_desktop, [95](#)
 - web_config_group, [96](#)
 - web_config_password, [97](#)
 - web_config_proxy, [98](#)
 - web_create_user, [99](#)
 - web_restart, [100](#)

web_root_password, [101](#)