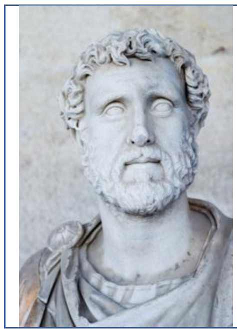


Epiphron Consulting Ltd

Dave Sinclair
das@epiphron-consulting.co.uk

DAF

A Distributed Automation Framework

v1.0 27 Oct 2011

DAF is an automation framework for executing test scenarios against software or hardware components in a system under test.

DAF

A Distributed Automation Framework

Table of Contents

Introduction

DAF is an automation framework for executing test scenarios against software or hardware components in a system under test. DAF can be applied to the testing of database servers, web servers, storage subsystems, storage area networks, information systems, client – server and peer to peer systems – in fact, DAF is applicable to any general configuration of computers that is being used to perform a set of tasks.

DAF has capabilities to:

- Manage a large number of test machines, including test hosts, switches, controllers, clusters etc Can sequence jobs on a single machine and terminate the sequence if a job fails
- Run multiple, concurrent jobs on various test hosts
- Capture and transfer the log output from each job to a central repository
- Keep an audit log of what tests were run, whether they passed or failed, what hardware configurations and software levels were used and which log files belong to which test.

Figure 1 on page 4 shows a generic DAF configuration. This consists of a number of test stands and a central DAF server. The DAF server machine contains an SQL database, a Web server and the DAF server itself – this is used to control the DAF agents in the individual test hosts in each test stand. Testers use a Web browser to communicate with the DAF server via the Web server. Typically a tester will ask the DAF server to run a test scenario on a particular test stand. To do this, the DAF server contacts the appropriate DAF agents in the test stand and these agents run the testcases associated with the scenario on the test hosts in the test stand. The results of each testcase are collected by each agent and communicated back to the DAF server where they are stored in a central SQL database. Any logs produced by the testcase (eg stdout, stderr) are copied from the agents to a repository at the central DAF server. If required, emails are sent to interested parties describing whether or not the scenario passed or failed. The tester uses a web browser to communicate with the DAF server and examine the outcome of the.

DAF can run multiple concurrent test scenarios on one or more test stands and new scenarios may be started while existing scenarios are already running. Existing scenarios may be cancelled before completion if desired.

Sequences of scenarios may be chained together, and choice of next scenario in the sequence can be changed according to the success or failure of the previous scenario.

Figure 2 on page 5 shows a more detailed view of a DAF configuration containing just a single Test Stand.

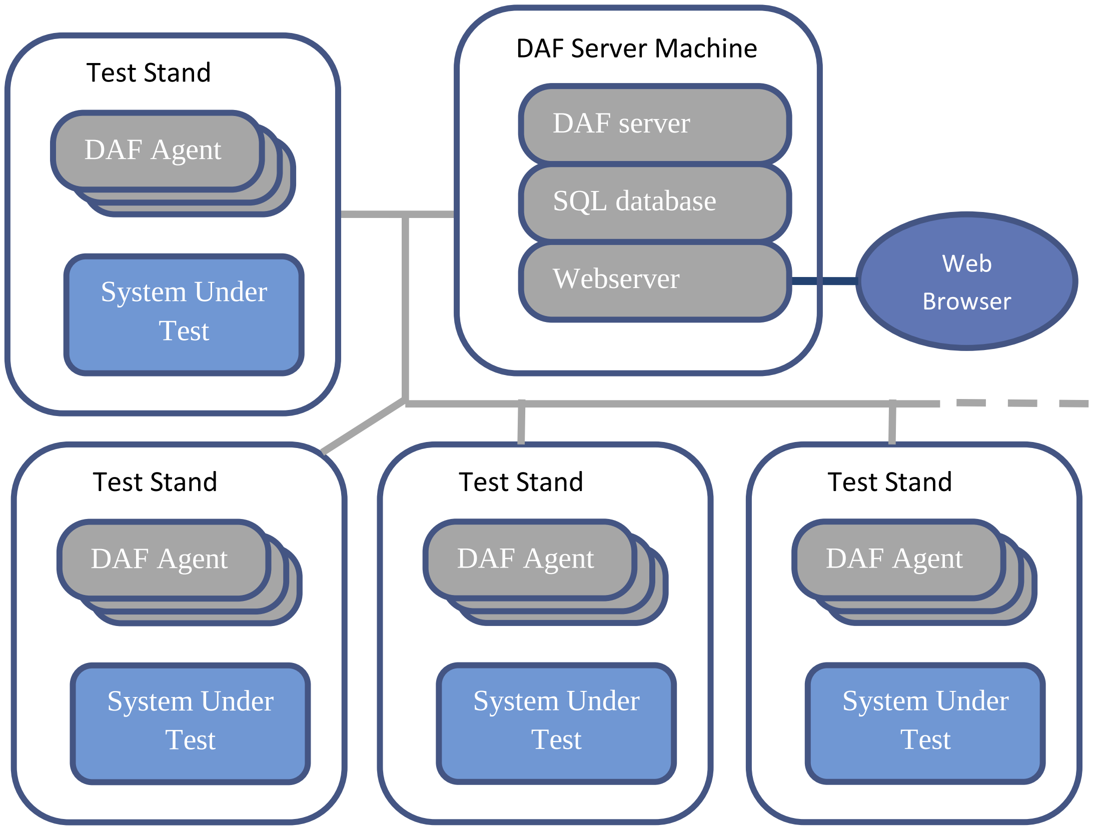


Figure 1 - Multiple Test Stand DAF Configuration

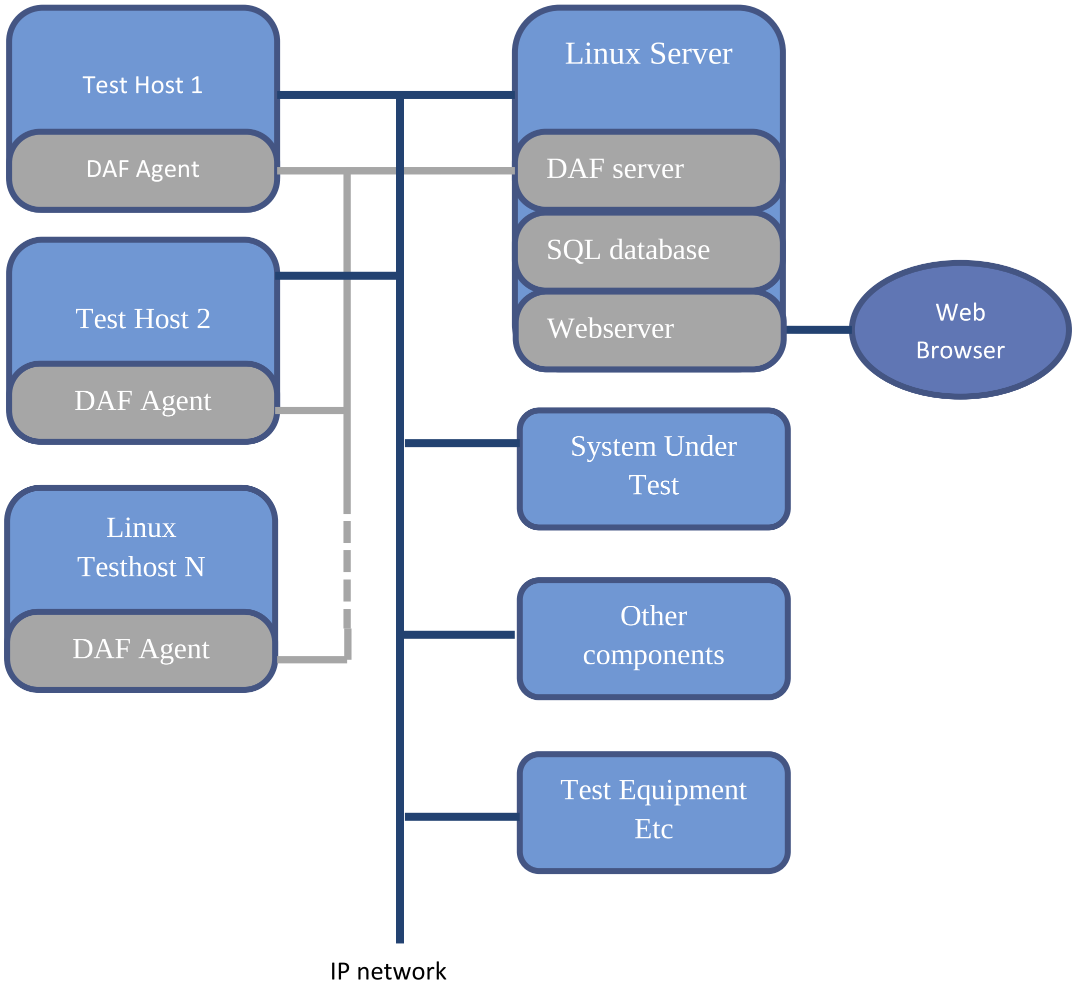


Figure 2- A DAF configuration containing just a single Test Stand.

DAF Concepts

The *Distributed Automation Framework (DAF)* consists of a set of DAF agents on each Test Host that are coordinated by the central DAF server. The Execution framework controls the running of each of the testcases within the scenario, and the error injection mechanisms etc. and collects the results from these operations.

A single DAF server may be used to test a variety of different systems under test. These might represent software or hardware products that are still in development, in which case testing would probably focus on functional, performance, reliability, serviceability, upgradeability and so on. A different system under test might also represent a product that is already being shipped to customers, in which case test scenarios might represent regression suites so that software fixes for the product could be tested before being shipped to customers. Another system might represent an actual customer configuration that corresponded to a customer problem report. DAF uses the concept of Products and Test Phases to allow the tester to classify a particular testing activity, and thus partition the DAF results into logical groups.

A *Product* refers to the name of the system under test. For example if Acme 32 port Fibre Channel Switch is being tested, then the Product name might be 'Acme32FC'. DAF is partitioned so that test scenarios and test results within one Product are separated from test scenarios and test results in other Products.

The test of a given product is normally divided into phases, e.g. functional testing, performance testing, system testing and so on. When a test is run by DAF it must be associated with a particular *Test Phase*. Test results are therefore grouped by test phase. Typical test phase names might be 'Acme32FC_FVT' and 'Acme32FC_SVT' for functional and system tests on the Acme Fibre Channel Switch.

DAF requires a tester to identify themselves before they can run a scenario. A DAF *User* is identified by their email address. Users must login to DAF, using a simple password mechanism. Generally any DAF user can perform any task and see any data within DAF. Each DAF server has special administrative user, which cannot be deleted, and which is used for any critical DAF tasks. A *Mail List* is a list of users. Typically a mail list is used to specify a group of users who are to be notified about a particular event, such as the outcome of a scenario.

A *Scenario* is a sequence of test operations. A scenario is composed of one or more *Steps*, and the steps are executed in sequence. Steps are numbered from 1 upwards. Each step may contain one or more *Actions*. An action consists of running a test script on a test host. All of the actions in a particular step are run at the same time – though if there are multiple actions in a step, they need not necessarily all run on the same test host.

A *Testcase* is a script or command that runs on a test host. The script will often represent a general test that is customized for different situations by a range of parameters. There is a one to one correspondence between an action in a scenario and testcase – that is to say, each action consists of running a single testcase on a single test host.

A *Level* is a measurable property of the code or hardware of the system under test. For example, if a Windows Server was being tested, a Level could be defined that represented the version of Windows running on the system under test. The value of the Level would then be something like "Windows 2003 R2". A *Level Instance* is combination of a particular level and a value of that level. For example, if a Linux server was under test, the development organization might produce several versions of Linux to test, each with a different build or kernel number. If a level was defined in DAF called LinuxKernel to represent these different builds, then if development produced 3 builds with kernels designated '2.6.37.1-1.2-desktop', '2.6.37.1-1.3-desktop' and '2.6.37.1-1.4-desktop', then this would define 3 separate level instances identified

as:
  LinuxKernel 2.6.37.1-1.2-desktop
  LinuxKernel 2.6.37.1-1.3-desktop
  LinuxKernel 2.6.37.1-1.4-desktop

A *Test Level* is a collection one or more level instances.   These define the code or hardware levels that must be present in the system under test before any scenario can be executed on that test stand.

A *Test Stand* is a collection of physical resources used to test the object under test. The object under test is considered to be part of the test stand. DAF agents are installed in Test Hosts within the Test Stand. This enables DAF to cause system commands, testcases or utilities to run on each test host and effectively gives DAF control over the behavior of each Test Host.   If it is necessary to control other objects in the test stand from within DAF, then this must be done by running a script or utility on a Test Host that communicates via a command line interface(CLI)/Web browser or other user interface on the object. For example, if DAF is perform configuration actions on say an IP network switch, then DAF might run SSH commands on a Test Host and these SSH commands would allow DAF to login to the IP network switch CLI and perform the desired configuration commands.

An *Object* is any physical resource within a teststand.   Typically these may be IP or SAN network switches, servers, storage devices, peripherals, test equipment or error injectors, however the tester may define any type of object they like. An object type has a name and a description, a single parent object type and optionally one or more child object types. The Test Stand is the parent object at the top of the object tree.

The *Environment* is the collection of environment variables seen by the process used by the DAF agent on a particular Test Host. An Environment would typically define the PATH environment variable and this would include the directory containing the Testcase, so that the process can find and execute the Testcase. Other environment variables may also be set, as a means of communicating parameters to the Testcase.

There are typically many test hosts in a test stand. It is necessary therefore to have a mechanism that allows the tester to specify which test host is to be used for a particular action. DAF provides two mechanisms for doing this – the *Host Selector* and the *Collector Set*.

Two types of host selector are supported:

Any	This is effectively the “Don’t care” selector and means that Host Selectors are not used in deciding which Test Host is to be used for the Action.
BySelectorValue	This means that any host with a specific Host Selector Value may be used in the Action

Each test host may be assigned a host selector value. This allows the tester to put test hosts into separate groups – typically the host selector value might reflect the usage or type of host, eg if hosts 1,2 and 3 are Windows hosts and hosts 4 and 5 are Linux hosts, then hosts 1-3 might be given the host selector value of ‘Windows’ and hosts 4-5 might be given the value “Linux”.   The tester could then use the host selector type of ‘BySelectorValue’ to specify that a particular action within a scenario must be executed on a test host with a host selector value of ‘Windows’. This would guarantee that DAF would select one of host 1, 2 or 3 to run that particular action.

A *Collector Type* is some aspect of a test host such as CPU architecture, installed memory size, clock speed, ip interface and so on. A collector process is run periodically that obtains the value of collector type on each test host. DAF provides a standard set of collector types that include Operation System type and version, installed DAF agent version, CPU architecture and installed RAM. The user may define additional collector types, by specifying a script or utility that is to be run on a test host and which provides the value of the collector.

A *Collector Set* is a collection of collector types and values which are used to determine if a particular test host should be used in an action. For example, a collector set might include collectors that indicate only test hosts with RAM > 1Gbyte and 64 bit CPUs are to be used to execute a particular action.

An *Outcome Action* determines what action should be taken if an individual action within a scenario fails, and what actions should be taken when a scenario ends. The tester may choose to stop the scenario on the first failed scenario action, or may choose to allow the scenario to run to completion, even if actions within the Scenario fail. The tester may also choose to stop execution at the end of the scenario or chain execution to a new scenario. Different actions can be specified if the scenario is successful or fails. The outcome action also specifies who, if anybody is to be notified (via email) if the scenario passes or fails.

The *Action Result* record in the DAF database records the details of the action and its outcome. This includes the test host used in the action; the command that was run and any associated parameters and whether the command or script passed or failed, as well as links to logs that contain the output of the command. The overall outcome of a scenario is recorded in a *Scenario Result*. This collects together all the action results from the scenario as well as any other relevant information (including the name of the tester who ran the scenario, the total number of actions attempted, passed, failed, the overall pass/fail result and pointers to additional logs describing the details of how the scenario was run). Logs from the actions and scenario are recorded in the *Log Repository* – this is simply a set of directories on the DAF server machine.

DAF uses Work Requests and a Work Queue to manage the execution of actions and scenarios. A *Work Request* is used to track the progress of each scenario while that scenario is executing. Once a scenario is complete, the work request is archived in the test results for the appropriate product and phase. This provides an audit trail of which scenarios have been executed, when they were executed and what their outcome was. Generally the tester will not need to look at this archive as the details of the result of each scenario are also recorded in the scenario result record. Items on the *Work Queue* are used to track the progress of each individual action within a scenario while that scenario is executing. Once a scenario is complete, the work queue items for that scenario are archived in the test results for the appropriate product and phase. This provides an audit trail of which scenario actions have been executed, when they were executed and what their outcome was. Generally the tester will not need to look at this archive as the details of the result of each scenario are also recorded in the scenario result record.

Teststand	
Name	FStesting
Comments	Used for filesystem testing

Test Hosts

Host Name	Type	Model	Serial	Host Selector Value	Primary Host in Test Stand?	Agent Status	Agent Status Date
suse11a	HP PC	A61452a	ZX-1323	group2	yes	Online	2011-10-17 18:06:41
toshiba	Laptop	Toshiba	ACX-43Q	group2		Online	2011-10-17 18:06:41
linux-dbw0	VM	Novatech	nov01-i7	group2		Online	2011-10-17 18:06:41
sparc1	Sparc	Sun X431	ZZ-TRE1	group2		Online	2011-10-17 18:06:41

NASbox

ID	Parent	Name	Comments
3	FStesting	QNAPTS410A	A QNAP NAS server

SATA Drives

ID	Parent	Name	Comments
1	QNAPTS410A	drive1	part of the QNAPTS410A NAS server
2	QNAPTS410A	drive2	part of the QNAPTS410A NAS server
3	QNAPTS410A	drive3	part of the QNAPTS410A NAS server

Figure 3 - An example test stand

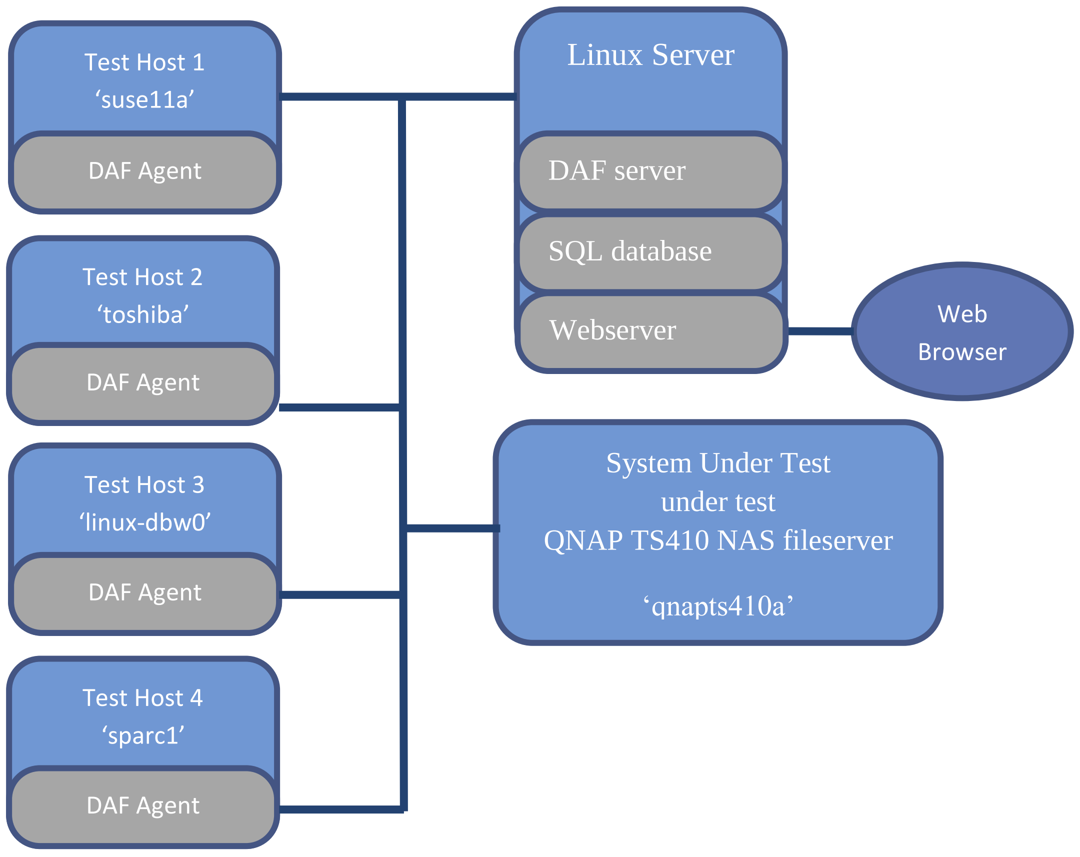


Figure 4 - An example test configuration

A DAF Example

To run a test, the tester must specify the three things:

- A test stand
- A test scenario
- A test level

The test stand

A simple test configuration is shown in Figure 5 on page 10. This test stand consists of four test hosts and one system under test. The test hosts are:

- 'suse11a' – an AMD Athlon 64 bit based desktop running SUSE 11 Linux
- 'toshiba' – an Intel Pentium 32 bit based laptop running Ubuntu Linux
- linux-dbw0 – a Linux Debian Virtual Machine running on a core-i7 Windows 7 server
- sparc1 – a Solaris based SPARC workstation

and the system under test is:

- 'qnapts410a' – a QNAP TS410 NAS server – this provides NFS exported filesystems that are accessed by the test hosts.

This configuration is being used to test the NFS filesystems exported by the TS410 NAS server. A filesystem exerciser program, nfst, will be used to perform read and write operations on the filesystems. nfst emulates an NFS client and reads and writes files from the NFS filesystems exported from the TS410 NAS server, performing directory and file creation and deletion and validating that that the data read back from the filesystems matches the data originally written. Each test host contains a directory, /testcases/nfst in which the 'nfst' NFS test exerciser program has been installed.

The Test Scenario

The test scenario is shown in Figure 6 on page 12.

The first step of the scenario contains 3 actions – one for each test host. Each action runs an instance of the nfst test exerciser on each test host. The first action uses the /testfs/data1 and /testfs/targ1 directories, the second action uses the /testfs/data2 and /testfs/targ2 directories and so on. Each instance of nfst performs wrc_test in using these directories. The wrc_test is a data integrity test where a number of data files are created and written in the specified filesystems and then the contents of these data files are validated.

Scenario	
Name	nfst_disk_pull
Description	Run a filesystem test from 3 hosts, then pull a disk from the RAID array, and reconfirm that the data written can be read and that the data is correct.
Outcome Action	Terminate

Steps									
Step Number	Description								
	Action	Collector set	Host Selector Type	Host Selector Value	Environment	Test case	Parameters	Duration	Duration Limit
1									
	RunTestcase In Environment		Use Primary Host		fst	nfst	-server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data1 -target_directory /testfs/targ1 -filesizes 1000 20000 4000000	Run until complete	
	RunTestcase In Environment	Anylinux	BySelector Value	group2	fst	nfst	-server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data2 -target_directory /testfs/targ2 -filesizes 1000 20000 4000000	Run until complete	
	RunTestcase In Environment	Anylinux	BySelector Value	group2	fst	nfst	-server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data3 -target_directory /testfs/targ3 -filesizes 1000 20000 4000000	Run until complete	
2									
	RunTestcase In Environment		Use Primary Host		fst	diskpull	-serial WD-WMAY02231295	Run until complete	
3									
	RunTestcase In Environment		Use Primary Host		fst	nfst	-server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data1 -target_directory /testfs/targ1 -filesizes 1000 20000 4000000	Run until complete	
	RunTestcase In Environment	Anylinux	BySelector Value	group2	fst	nfst	-server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data2 -target_directory /testfs/targ2 -filesizes 1000 20000 4000000	Run until complete	
	RunTestcase In Environment	Anylinux	BySelector Value	group2	fst	nfst	-server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data3 -target_directory /testfs/targ3 -filesizes 1000 20000 4000000	Run until complete	

Figure 6 - An example test scenario

The first action in step 1 is:

Action	Collector set	Host Selector Type	Host Selector Value	Environment	Test case	Parameters	Duration
RunTestcase In Environment		Use Primary Host		fst	nfst	-server qnapts410a -wrc_test -cycles 1 -export /testfs -filesizes 1000 20000 4000000 -data_directory /testfs/data1 -target_directory /testfs/targ1	Run until complete

The Host Selector Type is set to 'Use Primary Host' in this action. Referring to the test stand definition, it can be seen that this action will therefore run on the 'suse11a' host, as this is defined to be the primary host for this test stand. The action also states that the 'fst' environment should be used when running the 'nfst' testcase. This environment is defined as shown in Figure 7 on page 15. The DAF agent uses the fst environment to set up the PATH environment variable in the process that runs the nfst exerciser on the test host – and since the PATH contains the /testcases/nfst directory the process will therefore successfully locate this exerciser.

The second action in step 1 is:

Action	Collector set	Host Selector Type	Host Selector Value	Environment	Test case	Parameters	Duration
RunTestcase In Environment	Anylinux	BySelector Value	group2	fst	nfst_set	-server qnapts410a -wrc_test -cycles 1 -export /testfs -filesizes 1000 20000 4000000 -data_directory /testfs/data2 -target_directory /testfs/targ2	Run until complete

This action will not use the primary host, as the Host Selector Type is not set to 'Use Primary Host'. The selection mechanism in this case first looks at the Collector Values of each host to see if they match the Collector Set specified for the action. The effect of the Host Selector is then taken into account. Finally, if there are still multiple candidate hosts, the least busy host is selected.

The 'Anylinux' collector set specified here is shown in Figure 9 on page 15. This states that the P_OS_TYPE collector must have the value 'Linux' for any host that matches the collector set. In effect, this is saying that the host must be running some variant of the Linux operating system. Since the 'sparc1' host in the test stand is running Solaris, the collector set mechanism prevents this host being selected for this action.

The Host Selector Type in this action has been set to 'BySelectorValue' and the Host Selector Value has been set to 'group2'. Referring to the test host list in Figure 3 it can be seen that the candidate hosts for this action (ie those with Host Selector Values of 'group2' and not excluded by the collector set mechanism) are 'suse11a', 'toshiba', 'linux-dbw0'. DAF will select a host that is running the fewest actions in the Scenario, so in this case 'suse11a' will not be considered as it is already running the first action in the scenario. Consequently either 'toshiba' or 'linux-dbw0' will be used for this action. In this example it will be assumed that 'toshiba' is allocated to this action.

The third action in step 1 is:

Action	Collector	Host	Host	Environ	Test	Parameters	Duration

	set	Selector Type	Selector Value	ment	case		
RunTestcase In Environment		Use Primary Host		fst	nfst	-server qnapts410a -wrc_test -cycles 1 -export /testfs -filesizes 1000 20000 4000000 -data_directory /testfs/data3 -target_directory /testfs/targ3	Run until complete

Host selection in this action is very similar to the second action, again 'sparc1' is excluded by the collector set mechanism and 'suse11a', 'toshiba' and 'linux-dbw0' will be the candidate hosts but as 'suse11a' and 'toshiba' are already running actions, the third action will be given to 'linux-dbw0'.

The Duration field in each of the actions specifies that the testcase should be 'Run until complete' – in other words the action will continue running until the 'nfst' exerciser completes.

Once test hosts have been identified for each scenario action, DAF instructs the DAF agent in each test host to run the specified testcase, - which in this case is the 'nfst' file system exerciser. Each action will be deemed to pass if the 'nfst' exerciser returns good status - ie an exit code of 0. If a non-zero return code arises, the action will be marked as having failed in the DAF status database.

If an action fails within a step, and other actions are still running, these remaining actions are canceled immediately. The behavior of DAF in this case is determined by the value of 'Outcome Action' that is specified in the scenario. Referring to Figure 6 it can be seen that this has the value 'Terminate' in this example. This means that if an error occurs during an action, any other actions that are still running in that step are terminated as soon as possible and the execution of the scenario stops at the failing step. (DAF provides an alternate setting, 'ContinueOnError' which means that all actions within a scenario will be attempted, even if one or more them fails. This setting is used less frequently as it is generally desired that the system under test is stopped as soon as possible after an error occurs, so that debug information can be obtained while the system is in failure state).

If no errors occur in step 1 of the example scenario, then DAF proceeds to execute step 2 and so on. Step 2 in this case contains a single action:

Environments	
Name	fst
Description	An environment for file system testing

Environment Members	
Environment Variable Name	**Environment Variable Value**
PATH	/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/testcases:/testcases/nfst

Figure 7 - The fst environment

Collector Types			
Name	**Built in?**	**OStype**	**Invocation**
P_OS_VERSION	yes		
P_OS_TYPE	yes		
QNAPhardwarelevel		Linux	getqnaphwlevel.sh
QNAPsoftwarelevel		Linux	getqnapswlevel.sh

Figure 8 - Example Collector Types

Collector Set	
Name	Anylinux
Description	A collector set that picks out just the Linux hosts

Collector Set Members		
Collector Name	**Comparator**	**Collector Value**
P_OS_TYPE	EQ	Linux

Figure 9 - The Anylinux collector set

Host Collector Values		
Host Name	**Collector Name**	**Collector Value**
suse11a	P_OS_Type	Linux
suse11a	P_OS_VERSION	2.6.37.1-1.2-desktop
toshiba	P_OS_Type	Linux
toshiba	P_OS_VERSION	2.6.38-11-generic
linux-dbw0	P_OS_Type	Linux
linux-dbw0	P_OS_VERSION	2.6.37.1-1.2-desktop
sparc1	P_OS_Type	Solaris
sparc1	P_OS_VERSION	8.3.0.0

Figure 10 - Example Host Collector Values

Test Level		
Name	QNAP Test	
Description	Test Levels for QNAP filesystem testing	
Test Level Instance		
Parent Object Name	**Level Instance**	**Value**
NASbox	QNAPhwlevel	4.11
NASbox	QNAPswlevel	5.1.1

Figure 11 - An example Test Level

Levels				
Name	**Description**	**Associated Object Type**	**Collector Type**	**Proxy Collector Set**
QNAPhwlevel	Hardware level of a QNAP file server	NASbox	QNAPhardwarelevel	Anylinux
QNAPswlevel	Software level of a QNAP file server	NASbox	QNAPsoftwarelevel	Anylinux

Figure 12 - Example Levels

Action	Collector set	Host Selector Type	Host Selector Value	Environment	Test case	Parameters	Duration
RunTestcase In Environment		Use Primary Host		fst	diskpull	-serial WD-WMAY02231295	Run until complete

In this action, the diskpull testcase is run on 'suse11a', the primary test host in the test stand. The purpose of this testcase is to simulate the failure of a SATA disk drive in the QNAP TS410 enclosure. The arrangement for doing this is shown in Figure 13 on page 17. The diskpull testcase runs in the 'suse11a' host, and communicates (possibly via TCP/IP or a serial interface) with an electro-mechanical robot that can physically remove or replace the SATA disk drives in the TS410 enclosure. This simulates a disk failing and can be used to cause the RAID software in the TS410 to perform an hot spare takeover and an array rebuild operation.

Once the SATA disk has been disconnected from the QNAP enclosure in step 2 of the scenario, DAF proceeds to step 3. This is similar to step 1 of the scenario, and contains 3 actions that run the NFST file system exerciser again. The only difference between the actions in this step and step 1 is that the NFST file system exerciser is run in the 'read_compare' mode. This means that no new data is written to the QNAP filesystems, but instead the existing files in the specified filesystems are read and validated to confirm they contain the expected data patterns. The same host selection process is used in this step as in step 1.

Once the actions in step 3 have been completed, the scenario ends.

The Test Level

DAF provides a mechanism for validating that the hardware or software components under test in the test stand are at the intended levels. When a scenario is run, the tester must specify a Test Level which contains one or more definitions of

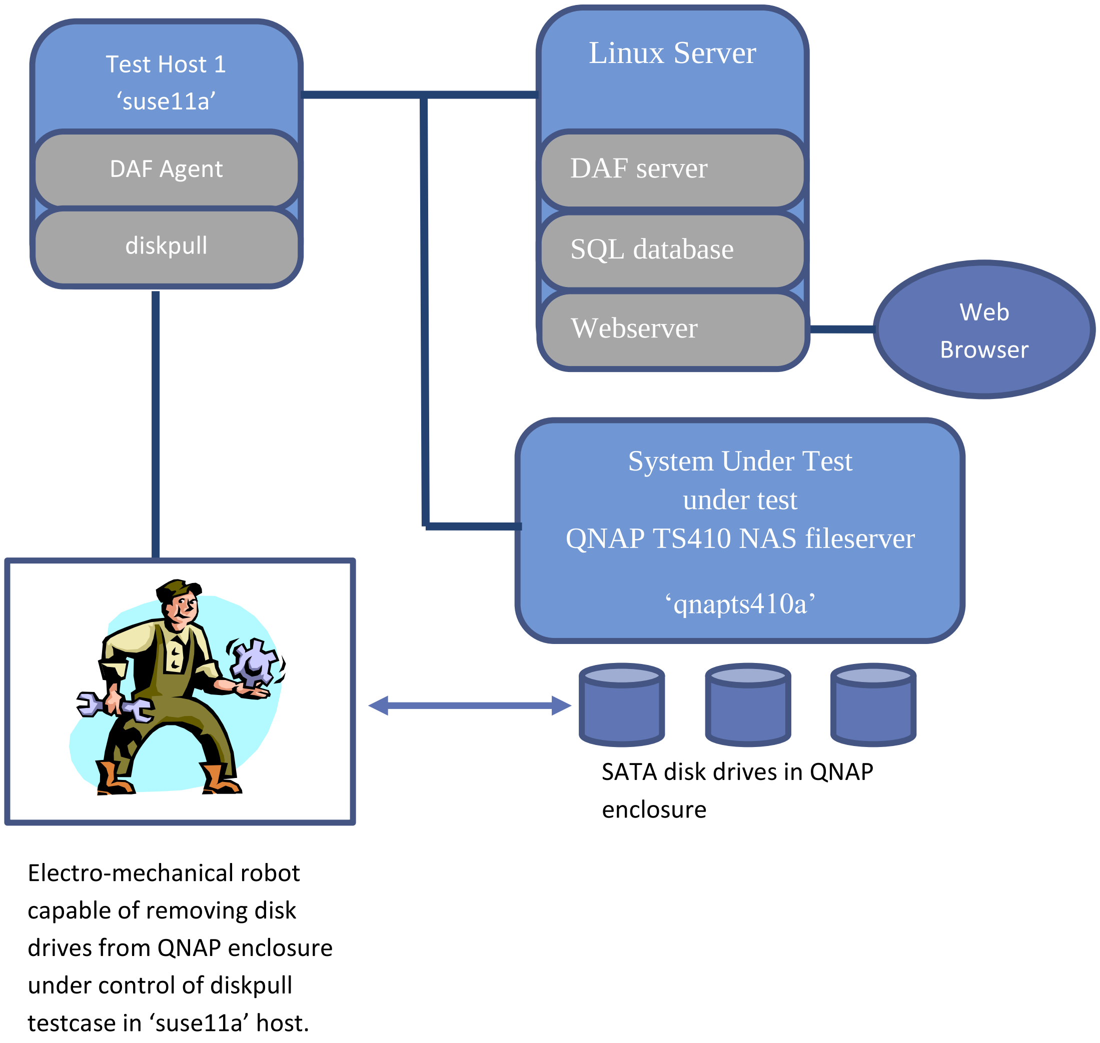
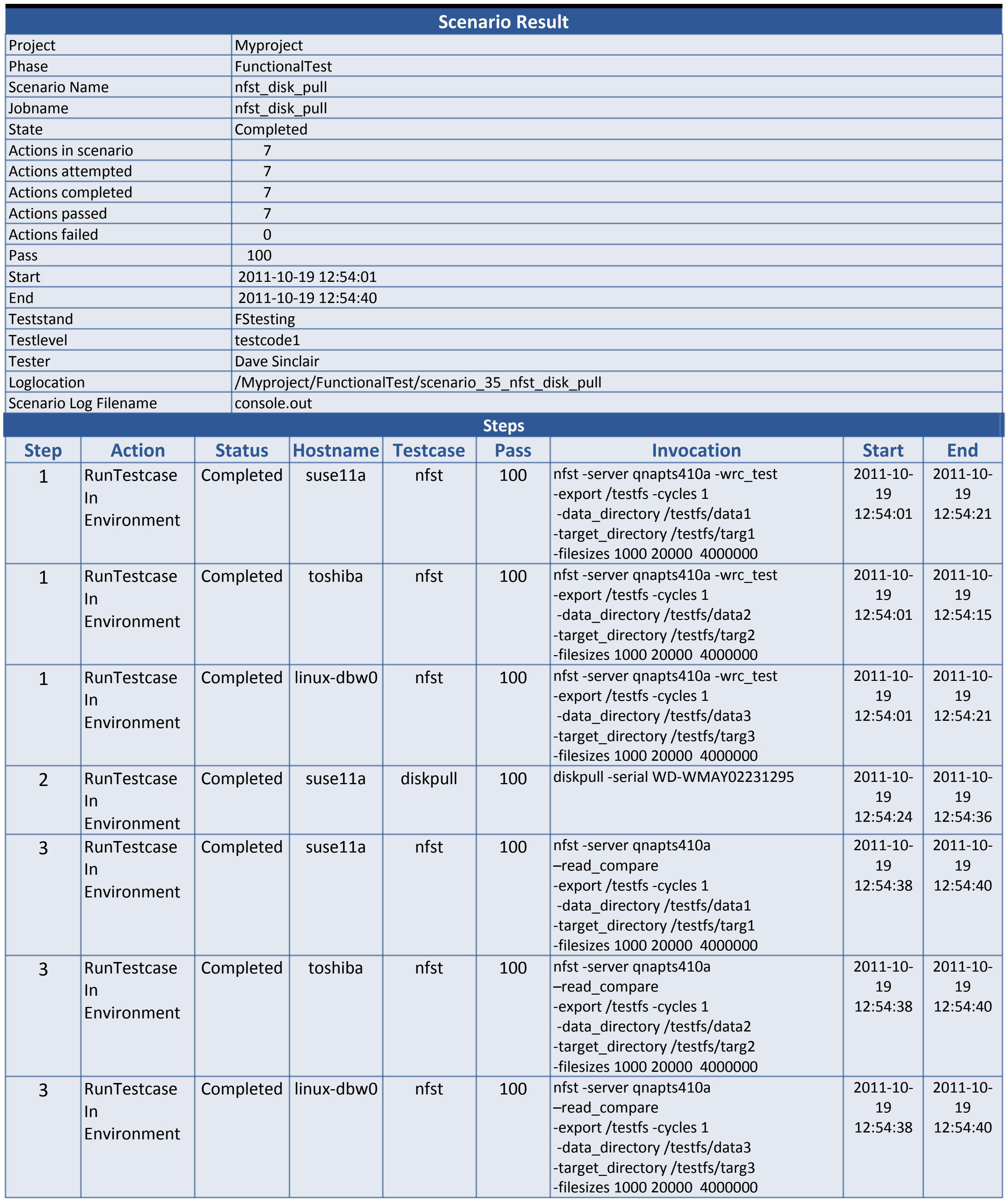

Test Host 1
'suse11a'
DAF Agent
diskpull
Linux Server
DAF server
SQL database
Webserver
Web Browser
System Under Test
under test
QNAP TS410 NAS fileserver
'qnapts410a'
SATA disk drives in QNAP enclosure
Electro-mechanical robot capable of removing disk drives from QNAP enclosure under control of diskpull testcase in 'suse11a' host.

Figure 13 - DAF control of an object that does not contain a DAF agent

hardware and software levels and the method by which these levels can be measured in the test stand. In this example, the levels of interest are the hardware version of the QNAP NAS server and the code level installed on this server. The Test Level that is used in this example is shown in Figure 11 on page 16. This shows that, if the scenario is to be allowed to run the QNAP TS410 must be at hardware level 4.11 and software level 5.1.1. DAF also needs to be told how to go about determining these values. To do this, DAF looks at the Level definitions provided by the tester. These are shown in Figure 11 on page 16. This shows that the value of the hardware level can be determined from the 'QNAPhardwarelevel' collector and that this can be obtained by running the collector on any host that matches the 'Anylinux' collector proxy set. Referring to Figure 9 it can be seen that the 'Anylinux' collector set would select any Linux host in the test stand and referring to Figure 8 it can be seen that the QNAPhwlevel level is determined using the QNAPhardwarelevel collector type. From Figure 8 it can be seen that QNAPhardwarelevel collector value is obtained by running the getqnaphwlevel.sh script. So to determine the hardware level on the QNAP system under test, DAF runs the getqnaphwlevel.sh on any Linux test host in the test stand and compares the result reported with the 4.11 specified in the Test Level (Figure 11). Similarly, the getqnapswlevel.sh script is run on any Linux test host to validate the QNAP software level.



Scenario Result	
Project	Myproject
Phase	FunctionalTest
Scenario Name	nfst_disk_pull
Jobname	nfst_disk_pull
State	Completed
Actions in scenario	7
Actions attempted	7
Actions completed	7
Actions passed	7
Actions failed	0
Pass	100
Start	2011-10-19 12:54:01
End	2011-10-19 12:54:40
Teststand	FStesting
Testlevel	testcode1
Tester	Dave Sinclair
Loglocation	/Myproject/FunctionalTest/scenario_35_nfst_disk_pull
Scenario Log Filename	console.out

Steps

Step	Action	Status	Hostname	Testcase	Pass	Invocation	Start	End
1	RunTestcase In Environment	Completed	suse11a	nfst	100	nfst -server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data1 -target_directory /testfs/targ1 -filesizes 1000 20000 4000000	2011-10-19 12:54:01	2011-10-19 12:54:21
1	RunTestcase In Environment	Completed	toshiba	nfst	100	nfst -server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data2 -target_directory /testfs/targ2 -filesizes 1000 20000 4000000	2011-10-19 12:54:01	2011-10-19 12:54:15
1	RunTestcase In Environment	Completed	linux-dbw0	nfst	100	nfst -server qnapts410a -wrc_test -export /testfs -cycles 1 -data_directory /testfs/data3 -target_directory /testfs/targ3 -filesizes 1000 20000 4000000	2011-10-19 12:54:01	2011-10-19 12:54:21
2	RunTestcase In Environment	Completed	suse11a	diskpull	100	diskpull -serial WD-WMAY02231295	2011-10-19 12:54:24	2011-10-19 12:54:36
3	RunTestcase In Environment	Completed	suse11a	nfst	100	nfst -server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data1 -target_directory /testfs/targ1 -filesizes 1000 20000 4000000	2011-10-19 12:54:38	2011-10-19 12:54:40
3	RunTestcase In Environment	Completed	toshiba	nfst	100	nfst -server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data2 -target_directory /testfs/targ2 -filesizes 1000 20000 4000000	2011-10-19 12:54:38	2011-10-19 12:54:40
3	RunTestcase In Environment	Completed	linux-dbw0	nfst	100	nfst -server qnapts410a –read_compare -export /testfs -cycles 1 -data_directory /testfs/data3 -target_directory /testfs/targ3 -filesizes 1000 20000 4000000	2011-10-19 12:54:38	2011-10-19 12:54:40

Figure 14 - An example Scenario Result

Recording Status

Status for each action in each step is recorded in the DAF status database as soon as each particular action completes. This status can be viewed in the Scenario Result table as shown in Figure 14 on page 18.

Job Status

When an action is run on a test host, a job control log is saved which records the details of the process used to run the testcase. The stdout and stderr streams from the process running the testcase of the action are also collected. All three logs are copied to the status repository for the test phase associated with the scenario that is running.

An example of a job control log is shown in Figure 15 on page 19. This corresponds to the first action in step 1 of the example scenario in Figure 6.

```
20111019 125401:                Preparing: tag=31898:  nfst -server qnapts410a -export /testfs …
0111019 125401:               execute_cmd: tag=31898, actionresultID=126
20111019 125401:            Update cmd_log: tag=31898:  tag=31898, pid=9397, setting state=3, status=0
20111019 125401:  Executing in environment: tag=31898:  nfst -server qnapts410a –export …
20111019 125401:               Environment: tag=31898:  PATH=/sbin:/usr/sbin:/usr/local/sbin:/…
20111019 125421:             zombie_reaper:  pid=9397 (CMD_COMPLETED), status=0 (exitcode=0, signal=0)
20111019 125421:             zombie_reaper:  nfst -server qnapts410a -export /testfs -data_directory…

Some lines have been truncated for clarity
```

Figure 15 - An example job control log

Each action in each step in the scenario will thus have a job control log, stdout and stderr log. There is also a single console log for the overall scenario. An example of this console log is shown in Figure 16 on page 20 and Figure 17 on page 21. The console log records:

- the names and IDs of the scenario, scenario result, tester, test stand and test levels
- whether or not the software and hardware levels in the test level specified when the scenario was started match the actual software and hardware levels on the test stand
- the proxy hosts that were used to determine the hardware and software code levels in the test stand
- the names and types of the objects were subjected to test level analysis
- the collector sets which were applied when determining which test hosts were used for each action in the scenario and the name of each test host used
- details of the outcome action that is to be applied (which determines what is to happen if the scenario passes or fails
- details of the maillist that is potentially used to inform testers of the outcome of the scenario

```
==============================================================================================
20111026 170642: Starting new Scenario
20111026 170642:    Project           = project1
20111026 170642:    Phase             = phase1
20111026 170642:    Scenario ID       = 70 (nfst_disk_pull)
20111026 170642:    ScenarioresultID  = 63
20111026 170642:    Testlevel ID      = 4 (QNAP Test)
20111026 170642:    Teststand ID      = 6 (FStesting)
20111026 170642:    OutcomeactionID   = 5
20111026 170642:    TesterID          = 2 (Yogi Bear (MrBear@hannabarbera.com))
20111026 170642:    MaillistID        = 2
20111026 170642:    Loglocation       = /tmp/log876/scenario_63_nfst_disk_pull
20111026 170642:    Scenario Log file = console.out
20111026 170642:
20111026 170642:    Outcomeaction Details:
20111026 170642:    Name:                  Terminate
20111026 170642:    Actiononpass:          None
20111026 170642:    Actiononfail:          None
20111026 170642:    Scenariotoberunonpass: 4 ()
20111026 170642:    Scenariotoberunonfail: 52 ()
20111026 170642:    Erroraction:           Terminatescenario
20111026 170642:    Notificationonpass:    Tester
20111026 170642:    Notificationonfail:    Tester
20111026 170642:    TesterID:              0 (  ())
20111026 170642:    MaillistID:            2 (another)
----------------------------------------------------------------------------------------------
20111026 170642: Validating object code levels in TeststandID 6 (FStesting) against Testlevel 4 (QNAP Test)
20111026 170642:    Testlevelmember: levelname = QNAPhwlevel, objecttype = NASbox, desiredvalue = 4.11
20111026 170642:    Required collectorvalues are:
20111026 170642:       Collectorset           Name               Comparator   Required Value
20111026 170642:       2.6.37.1-1.2-desktop   P_OS_VERSION       EQ           2.6.37.1-1.2-desktop
20111026 170642:       suse11a met the host proxy collector requirements for this object
20111026 170642:       Collectortype (QNAPhardwarelevel) collectorvalue (4.11) from object QNAPTS410A matches
                       required value 4.11
20111026 170642:    Testlevelmember: levelname = QNAPswlevel, objecttype = NASbox, desiredvalue = 5.1.2
20111026 170642:    Required collectorvalues are:
20111026 170642:       Collectorset           Name               Comparator   Required Value
20111026 170642:       2.6.37.1-1.2-desktop   P_OS_VERSION       EQ           2.6.37.1-1.2-desktop
20111026 170642:       suse11a met the host proxy collector requirements for this object
20111026 170642:       Collectortype (QNAPsoftwarelevel) collectorvalue (5.1.2) from object QNAPTS410A matches
                       required value 5.1.2
```

Figure 16 - An example Scenario console log (part 1)

```
20111026 170642: Preparing action in step: 1  hostname: suse11a  workqueue.ID: 89564  Maxduration: 60 secs  Duration: Rununtilcomplete
20111026 170642: Executing action in step: 1  hostname: suse11a  workqueue.ID: 89564  tag: 92  Maxduration: 60 secs  Duration: Rununtilcomplete
20111026 170642:    Environment[0]:
PATH=/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/testcases/nfst
20111026 170642:    Invocation: nfst -server qnapts410a -export /testfs -data_directory /testfs/data1 -target_directory /testfs/target1
                                 -filesizes 10000 200000 4000000 -wrc_test -cycles 1 -num_directories 10 -cycles 1
20111026 170642: Preparing action in step: 1  hostname: toshiba  workqueue.ID: 89565  Maxduration: 60 secs  Duration: Rununtilcomplete
20111026 170642: Executing action in step: 1  hostname: toshiba  workqueue.ID: 89565  tag: 68  Maxduration: 60 secs  Duration: Rununtilcomplete
20111026 170642:    Environment[0]:
PATH=/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/testcases/nfst
20111026 170642:    Invocation: nfst -server qnapts410a -export /testfs -data_directory /testfs/data2 -target_directory /testfs/target2
                                 -filesizes 10000 200000 4000000 -wrc_test -cycles 1 -num_directories 10 -cycles 1
20111026 170642: Preparing action in step: 1 hostname: linux-dbw0  workqueue.ID: 89566 Maxduration: 60 secs Duration: Rununtilcomplete
20111026 170642: Executing action in step: 1 hostname: linux-dbw0  workqueue.ID: 89566 tag: 92  Maxduration: 60 secs Duration: Rununtilcomplete
20111026 170642:    Environment[0]:
PATH=/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin::/testcases:/testcases/nfst
20111026 170642:    Invocation: nfst -server qnapts410a -export /testfs -data_directory /testfs/data3 -target_directory /testfs/target3
                                 -filesizes 10000 200000 4000000 -wrc_test -cycles 1 -num_directories 10 -cycles 1
20111026 170644:    Completed action in step 1, hostname toshiba, workqueue.ID 89565, pass 100
20111026 170645:    Copied /tmp/log876/scenario_63_nfst_disk_pull/agent_log_toshiba_381.log from test host toshiba, log directory
                           /opt/daf/logs/agent_log_toshiba_381.log
etc
etc
20111026 170705:    Completed action in step 1, hostname linux-dbw0, workqueue.ID 89566, pass 100
20111026 170705:    Copied /tmp/log876/scenario_63_nfst_disk_pull/daf_agent.log from test host linux-dbw0, log directory
                           /opt/daf/logs/daf_agent.log
etc
etc
20111026 170701:    Completed action in step 1, hostname suse11a, workqueue.ID 89564, pass 100
20111026 170701:    Copied /tmp/log876/scenario_63_nfst_disk_pull/suse11a_380_nfst.stdout from test host suse11a, log directory
                           /opt/daf/logs/suse11a_380_nfst.stdout

20111026 170705: Completed all actions in step: 1  ScenarioresultID: 63  terminate_scenario: 0
=========================================================================================================
```

Figure 17 - An example Scenario console log (part 2)

DAF Reference Information

Test Stand

A Test Stand is a collection of physical resources used to test the object under test. The object under test is considered to be part of the test stand. DAF agents are installed in Test Hosts within the Test Stand. This enables DAF to cause system commands, testcases or utilities to run on each test host and effectively gives DAF control over the behavior of each Test Host. If it is necessary to control other objects in the test stand from within DAF, then this must be done by running a script or utility on a Test Host that communicates via a command line interface (CLI)/Web browser or other user interface on the object. For example, if DAF is perform configuration actions on say an IP network switch, then DAF might run SSH commands on a Test Host and these SSH commands would allow DAF to login to the IP network switch CLI and perform the desired configuration commands.

Test Host

There are typically many Test Hosts in a Test Stand. It is necessary therefore to have a mechanism that allows the tester to specify which Test Host is to be used for a particular Action. DAF provides two mechanisms for doing this – the Host Selector and the Collector Set. Two types of Host Selector are supported:

Any	This is effectively the "Don't care" selector and means that Host Selectors are not used in deciding which Test Host is to be used for the Action.
BySelectorValue	This means that any host with a specific Host Selector Value may be used in the Action

Each Test Host may be assigned a Host Selector Value. This allows the tester to put Test Hosts into separate groups – typically the Host Selector Value might reflect the usage or type of host, eg if hosts 1,2 and 3 are Windows hosts and hosts 4 and 5 are Linux hosts, then hosts 1-3 might be given the Host Selector Value of 'Windows' and hosts 4-5 might be given the value "Linux". The tester could then use the Host Selector Type of 'BySelectorValue' to specify that a particular Action within a Scenario must be executed on a Test Host with a Host Selector Value of 'Windows'. This would guarantee that DAF would select one of host 1,2 or 3 to run that particular action.

Collector Type and Value

A Collector Type is some aspect of a Test Host such as CPU architecture, installed memory size, clock speed, ip interface and so on. A Collector process is run periodically that obtains the value of Collector Type on each Test Host. DAF provides a standard set of Collector Types that include Operation System type and version, installed DAF agent version, CPU architecture and installed RAM. The user may define additional Collector Types, by specifying a script or utility that is to be run on a Test Host and which provides the value of the Collector.

Standard properties

Property name	**Property type**	**Description**	**Valid values**
P_NAME	Built in	The name of the host	
P_CPU_ARCHITECTURE	Built in	The CPU architecture of the	x86

		host	x86_64 ia64 PowerPC sparc unknown
P_INSTALLED_RAM	Built in	The amount of main RAM memory installed in the host in Mbytes	0-N
P_OSTYPE	Built in	The Operating System installed on the host	AIX Linux Solaris HPUX Windows unknown
P_OS_SUPPLIER	Built in	The Operation System vendor or supplier	IBM Suse OpenSuse Redhat Ubuntu Debian Microsoft unknown
P_OS_VERSION	Built in	The complete version string for the OS	this will be OS dependendent eg AIX 5.1.3 eg. Suse 11.4.0.1
P_OS_MAJOR	Built in	The major part of the OS version, ie the 11 in 11.4.0.1	N
P_OS_MINOR	Built in	The minor part of the OS version, ie the 4 in 11.4.0.1	N
P_OS_Description	Built in		A free format description

User specified properties

In addition the user may specify collectors for any property that can be determined by running a program or script on the host.   To do this, the user must specify the name of the collector and the script/program that is to be run.   The script must follow these rules:

1. If the script/program fails, it must exit with a non zero return code. In this event, it is also helpful for debug if the script produces a message indicating the cause of the failure – this message should appear on STDOUT.
2. If the script/program succeeds, it must exit with a zero return code and it should print the value of the property to STDOUT.   Multiple line properties are supported but should be avoided if possible. The script should produce no other output.
3. Collector scripts are run in the shell environment of the DAF agent.   That is to say they inherit the PATH and environment of the process that the DAF agent was started in.

The following is an example of a collector that captures the size of the / filesystem on a Unix type host

Property name	Property type	Description	Valid values
ROOTFILESYSTEMSIZE	User defined	The size of the root filesystem on a Unix system, in Mbytes	0-N

The collector script would be

```
df  -BM / | grep -v Filesystem  | awk {'print $2'} | sed -e 's/M//'
```

User specified collectors are unlikely to apply to all operating system types. For instance, the ROOTFILESYSTEMSIZE collector above will only work on AIX, Linux, Solaris and HPUX operating systems and not on Novell, Windows or Mac. User specified collectors therefore include a list of the operation systems types that the collector should be applied to. The full definition of the ROOTFILESYSTEMSIZE collector is therefore:

Property name	Property type	Description	Valid values	Applicable OS
ROOTFILESYSTEMSIZE	User defined	The size of the root filesystem on a Unix system, in Mbytes	0-N	AIX, Linux, Solaris, HPUX

Object

An object is any physical resource within a teststand. Typically these may be IP or SAN network switches, servers, storage devices, peripherals, test equipment or error injectors, however the tester may define any type of object they like. An object type has a name and a description, a single parent object type and optionally one or more child object types. The Test Stand is the parent object at the top of the object tree.

Object Type

An object is any physical resource within a teststand. Typically these may be IP or SAN network switches, servers, storage devices, peripherals, test equipment or error injectors; however the tester may define any type of object they like. An object type has a name and a description, a single parent object type and optionally one or more child object types. The Test Stand is the parent object at the top of the object tree.

Scenarios

A scenario is a sequence of test operations and is made up of steps and actions. A scenario also contains an outcome action which defines what actions are to be taken according to whether the scenario passes or fails.

Step

A scenario is composed of one or more steps, and the steps are executed in sequence. Steps are numbered from 1 upwards. Each step may contain one or more actions.

Action

An action consists of running a test script on a test host. All of the actions in a particular step are run at the same time – though if there are multiple actions in a step, they need not necessarily all run on the same test host.

Environment

The Environment is the collection of environment variables seen by the process used by the DAF agent to run an action on a particular Test Host. An Environment would typically define the PATH environment variable and this would include the directory containing the Testcase, so that the process can find and execute the Testcase. Other environment variables may also be set, as a means of communicating parameters to the Testcase.

Collector Set

A Collector Set is a collection of Collector Types and Values which are used to determine if a particular Test Host should be used in an Action. For example, a Collector Set might include Collectors that indicate only Test Hosts with RAM > 1Gbyte and 64 bit CPUs are to be used to execute a particular Action.

Outcome Action

An Outcome Action determines what action should be taken if an individual Action within a Scenario fails, and what actions should be taken when a Scenario ends. The tester may choose to stop the Scenario on the first failed Scenario Action, or may choose to allow the Scenario to run to completion, even if Actions within the Scenario fail. The tester may also choose to stop execution at the end of the Scenario or chain execution to a new Scenario. Different actions can be specified if the Scenario is successful or fails. The Outcome Action also specifies who, if anybody is to be notified (via email) if the Scenario passes or fails.

Testcase

A Testcase is a script or command that runs on a test host. The script will often represent a general test would be customized for different situations by a range of parameters. There is a one to one correspondence between an Action in a Scenario and Testcase – that is to say, each Action consists of running a Testcase on a Test Host.

Level

A Level is a measurable property of the code or hardware of the system under test. For example, if a Windows Server was being tested, a Level could be defined that represented the version of Windows running on the system under test. The value of the Level would then be something like “Windows 2003 R2”.

Level Instance

A Level Instance is combination of a particular Level and a value of that level. For example, if a Linux server was under test, the development organization might produce several versions of Linux to test, each with a different build or kernel number. If a Level was defined in DAF called LinuxKernel to represent these different builds, then if development produced 3 builds with kernels designated ‘2.6.37.1-1.2-desktop’, ‘2.6.37.1-1.3-desktop’ and ‘2.6.37.1-1.4-desktop’, then this would define 3 separate Level Instances identified as:
LinuxKernel 2.6.37.1-1.2-desktop

LinuxKernel 2.6.37.1-1.3-desktop
LinuxKernel 2.6.37.1-1.4-desktop

The purpose of a Level Instance is to identify a code or hardware level that must be found on the system under test when a Scenario is run. Thus the tester might indicate that Scenarios X and Y should be run on Teststands A and B with Level Instance LinuxKernel 2.6.37.1-1.3-desktop. DAF will verify that this level is indeed present on Teststands A and B before allowing Scenarios A and B to execute on these test stands.
When a Scenario's results are recorded by DAF, they include the Level Instances that were present on the Test Stand during the test. This allows the user to be able to produce DAF reports that show Scenario success or failure rates on each Level Instance.

Test Level

A Test Level is a collection one or more Level Instances. These define the code or hardware levels that must be present in the system under test before any Scenario can be executed on that test stand. See Level Instance.

Collector Sets

Suppose we only wish to run tests on hosts that Suse Linux hosts that are running version 11 of OpenSuse and a RAM size of at least 1G. The Collector Set that defines a suitable host would be:

P_OSTYPE = OpenSuse
P_OS_MAJOR = 11
P_INSTALLED_RAM > 1023

Scenario Results

The overall outcome of a scenario is recorded in a Scenario Result. The Scenario Result record collects together all the action results from the scenario as well as any other relevant information (including the name of the tester who ran the scenario, the total number of actions attempted, passed, failed, the overall pass/fail result and pointers to additional logs describing the details of how the scenario was run). Logs from the actions and scenario are recorded in the Log Repository – this is simply a set of directories on the DAF server machine.

Action Results

The Action Result record in the DAF database records the details of the action and its outcome. This includes the test host used in the action; the command that was run and any associated parameters and whether the command or script passed or failed, as well as links to logs that contain the output of the command.

Users

DAF requires a tester to identify themselves before they can run a scenario. A DAF *User* is identified by their email address. Users must login to DAF, using a simple password mechanism. Generally any DAF user can perform any task and see any data within DAF. Each DAF server has special administrative user, which cannot be deleted, and which is used for any critical DAF tasks. A *Mail List* is a list of users. Typically a mail list is used to specify a group of users who are to be notified about a particular event, such as the outcome of a scenario.

Maillist

A list of users – typically a Mail List is used in an Outcome Action to specify a group of users who are to be notified about the outcome of a Scenario.

Host Selection in a Scenario

As an example, consider a test stand called FStesting that contains 4 test hosts:

Test Hosts								
Name	**Test Stand**	**Type**	**Model**	**Serial**	**Host Selector Value**	**Primary Host in Test Stand?**	**Agent Status**	**Agent Status Date**
suse11a	FStesting	HP PC	A61452a	ZX-13234		yes	Online	2011-10-14 03:24:59
toshiba	FStesting	Laptop	Toshiba	ACX-43Q	group2		Agent Offline	2011-10-14 03:25:10
linux-dbw0	FStesting	VM	Novatech	novatech-i7	group2		Online	2011-10-17 18:06:41
debian1	FStesting	AMD PC	X2345	88-4A3	group1		Online	2011-10-17 18:06:50

The primary host in this test stand is 'suse11a'. This is a choice that is made by the tester when the details of the test stand are entered into DAF. The tester may subsequently assign primary host status to a different host in the test stand, but there may be only one primary host in the test stand at any one time. Each host in the test stand (apart from the primary host) is assigned a Host Selector Value. This value may be used to divide the hosts into subsets. Here there are two hosts in the 'group2' subset and a further host in 'group1'.

When a Scenario is executed on this test stand, each Action within the Scenario will be executed on a particular test host. The process used to select the host that is used for a particular will be illustrated using the following Scenario Step:

Step.Action	Action	Host Selector Type	Host Selector Value	Collector set	Environment	Testcase	Parameters
1.1	RunTestcase InEnvironment	UsePrimary Host	group2	linux1	nfst_env	nfst	-server qnapts410a …
1.2	RunTestcase InEnvironment	BySelector Value	group2	linux1	nfst_env	nfst	-server qnapts410a …
1.3	RunTestcase InEnvironment	Any	group2	linux1	nfst_env	nfst	-server qnapts410a …

At the start of each action, DAF uses the the following process to select the host used to run that action:

1. Get a list of all the hosts that are allocated to the test stand that is being used in this scenario
2. If the Host Selector Type is set to 'UsePrimaryHost', select the Primary Host in the test stand – selection is then complete.

3. If the Host Selector Type is not set to ‘UsePrimaryHost’, then DAF examines the collectorset for this step and eliminates any hosts that do not have the desired collectorvalues – the remaining hosts are called the *candidate hosts* for this action.
4. If the Host Selector Type is set to ‘BySelectorValue’ DAF then removes any hosts in from the candidate host list that do not have a Host Selector Value that matches the Host Selector Value specified in the Scenario Action.
5. At this point, any remaining hosts in the candidate host list are regarded as valid hosts for this action.
6. If there is more than one host in the remaining candidate host list, the number of actions each host is already running in this Scenario Step will be taken into account, and the host that is running the lowest number of actions at that time will be selected. If more than one host meets this criterion, DAF selects a host from this group arbitrarily (but not randomly).

This selection process has the following consequences:

- If the Scenario specifies that an action is to be run on the Primary Host, then that action will always run on that specific host, independent of what other actions are already running on that host.
- Load balancing across hosts is done on a per action basis – if different actions produce different CPU loadings (eg if several actions are compute intensive testcases while other testcases are low CPU intensity, i/o testcases), then this different CPU loading will not be taken account of in the DAF host allocation – and it is probable that different hosts will have significantly different CPU loadings during this scenario step.
- Load balancing is only done within a given Scenario Step. If the same test stand is running two concurrent Scenarios, host loading by one Scenario will not affect the host selection process used in the other Scenario.

Environment Selection on Test Hosts

The Environment is the collection of environment variables present in the process used to run a testcase on a remote Test. An Environment would typically define the PATH environment variable and this would include the directory containing the Testcase, so that the process can find and execute the Testcase. Other environment variables may also be set, as a means of communicating parameters to the Testcase.

An Environment must be specified for each Action in every Step in a Scenario. Typically though, the same environment is used throughout a Scenario. The most common situation in which different Environments are needed is when different types of test host (eg Linux, Solaris) are being used for different steps in the scenario.

Consider the following example:

Environments	
Environment Name	**Description**
normal	A standard environment for testcases
fst	An environment for file system testing

Here two different environments have been defined. The details of the first environment are:

Environments	
Name	normal
Description	A standard environment for testcases
Environment Members	
Environment Variable Name	**Environment Variable Value**
PATH	/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib64/jvm/jre/bin:/testcases

This contains a single member, which defines the value of the PATH environment variable. In this example, the value of this variable includes the “/testcases” directory. The intent here is that this directory is present on each test host and contains all the scripts or programs to be run during the test.

The details of the second environment are:

Environments	
Name	fst
Description	An environment for file system testing
Environment Members	
Environment Variable Name	**Environment Variable Value**
PATH	/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/testcases:/testcases/nfst
FILESYSTEM_ROOT	/testfs
FILESYSTEM_TEST_PARM1	100
FILESYSTEM_TEST_PARM2	200

This not only contains a member that defines the path to the “/testcases” directory, but also contains a path to “/testcases/nfst”. This is because the environment is for use in filesystem testing and the tester has placed the NFST test program in the /testcases/nfst directory on each host. Additional environment variables have also been defined (FILESYSTEM_ROOT etc) which may be used by the scripts that are being run as part of the filesystem test scenarios.

Installing DAF

Pre-requisites

The DAF server may be installed on a Linux distribution. A 64 or 32 bit distribution may be used, however if a 64 bit distribution in use, then the 32 bit compatability C libraries must be available. For instance it may be necessary to install the ia32-libs package, eg on Ubuntu:

```
sudo apt-get install ia32-libs
```

As DAF uses RPC for inter host communication, the portmap service must be running on both the DAF server and any DAF agents. For instance it may be necessary to install the portmap package, eg on Ubuntu:

```
sudo apt-get install portmap
```

Installing the DAF server

1. Copy the daf_server.tar.gz package to a temporary directory on the host that will run the DAF server
2. Unpack the daf_server.tar.gz package using:

   ```
   tar –zxvf  daf_server.tar.gz
   ```

3. Run the

   ```
   install_daf_server
   ```

   script. This installs an apache web server on port 8001 and a mysql server on port 8002. The daf_server executable is installed in /opt/daf_server and will be started automatically by this script.

Installing the DAF agent

1. Copy the daf_agent.tar.gz package to a temporary directory on the host that will run the DAF server
2. Unpack the daf_agent.tar.gz package using:

   ```
   tar –zxvf  daf_agent.tar.gz
   ```

3. Run the

   ```
   install_daf_agent
   ```

   script. The daf_agent executable is installed in /opt/daf_agent and will be started automatically by this script.

Running DAF

Running the DAF server

The DAF server may be installed or upgraded manually:

```
daf -service install
```

The install option checks to see if an existing daemon is running. If it is, it is terminated.

When daf is installed onto a machine the executable is copied into /opt/daf. The copy of the daf executable that has been invoked is copied to to/opt/daf (creating this directories if necessary).

On Unix based machines (AIX, Linux, Solaris, HPUX) the daf installation modifies the /etc/inittab file so that /opt/daf/daf is started as a daemon when the machine is booted.   A daemon process is also launched immediately so the service can be used without having to reboot the machine. (This latter approach is not really recommended though as the environment in which the daf daemon then runs is determined by the environment in which daf is being installed, rather than the standard environment set up at boot time).

The daf service daemon can be stopped manually as follows:

```
daf -service stop
```

The daemon can be removed from the test host using

```
daf -service delete
```

This stops any daemon that is currently running, and removes any daemon entry for daf from the /etc/inittab file.

The full range of daf service parameters are

```
daf –service start | stop | delete | install | run | -consolelog <pathname>
```

The start | stop | delete | install | run can all be run in the same invocation. The –consolelog indicates the pathname of the file where the console log from the daf service is saved. This is typically /opt/daf/logs/daf.log.

Running the DAF agent

The DAF server may be installed or upgraded manually:

```
daf_agent -service install
```

The install option checks to see if an existing daemon is running. If it is, it is terminated.

When daf_agent is installed onto a machine the executable is copied into /opt/daf_agent. The copy of the daf_agent executable that has been invoked is copied to to/opt/daf_agent (creating this directory if necessary).

On Unix based machines (AIX, Linux, Solaris, HPUX) the daf_agent installation modifies the /etc/inittab file so that /opt/daf_agent/daf_agent is started as a daemon when the machine is booted. A daemon process is also launched immediately so the service can be used without having to reboot the machine. (This latter approach is not really recommended though as the environment in which the daf_agent daemon then runs is determined by the environment in which daf_agent is being installed, rather than the standard environment set up at boot time).

The daf_agent service daemon can be stopped manually as follows:

```
daf_agent -service stop
```

The daemon can be removed from the test host using

```
daf_agent -service delete
```

This stops any daemon that is currently running, and removes any daemon entry for daf_agent from the /etc/inittab file.

The full range of daf_agent service parameters are

```
daf_agent –service start | stop | delete | install | run | -consolelog <pathname>
```

The start | stop | delete | install | run can all be run in the same invocation. The –consolelog indicates the pathname of the file where the console log from the daf service is saved. This is typically /opt/daf_agent/logs/daf_agent.log.

Uninstalling DAF

Uninstalling the DAF server

1. Run the

   /opt/uninstall_daf_server

   script. This removes the DAF apache web, DAF mysql server and daf_server executable from this host. All test results that were stored in this DAF test repository are deleted.

Uninstalling the DAF agent

1. Run the

   /opt/uninstall_daf_agent

   script. This removes the daf_agent executable from this test host.