|  |  |
| --- | --- |
|  | Epiphron Consulting Ltd  Dave Sinclair [das@epiphron-consulting.co.uk](mailto:das@epiphron-consulting.co.uk) |

|  |
| --- |
| DAF |
| A Distributed Automation Framework - Architecture |
| v1.01 11 Dec 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 - Architecture

Table of Contents

[Introduction 3](#_Toc311402414)

[DAF Concepts 6](#_Toc311402415)

[A DAF Example 11](#_Toc311402416)

[The test stand 11](#_Toc311402417)

[The Test Scenario 11](#_Toc311402418)

[The Test Level 16](#_Toc311402419)

[Recording Status 19](#_Toc311402420)

[Job Status 19](#_Toc311402421)

[21](#_Toc311402422)

[DAF Reference Information 22](#_Toc311402423)

[Test Stand 22](#_Toc311402424)

[Test Host 22](#_Toc311402425)

[Collector Type and Value 22](#_Toc311402426)

[Standard properties 22](#_Toc311402427)

[User specified properties 23](#_Toc311402428)

[Object 24](#_Toc311402429)

[Object Type 24](#_Toc311402430)

[Scenarios 24](#_Toc311402431)

[Step 24](#_Toc311402432)

[Action 25](#_Toc311402433)

[Environment 25](#_Toc311402434)

[Collector Set 25](#_Toc311402435)

[Outcome Action 25](#_Toc311402436)

[Testcase 25](#_Toc311402437)

[Level 25](#_Toc311402438)

[Level Instance 25](#_Toc311402439)

[Test Level 26](#_Toc311402440)

[Collector Sets 26](#_Toc311402441)

[Scenario Results 26](#_Toc311402442)

[Action Results 26](#_Toc311402443)

[Users 26](#_Toc311402444)

[Maillist 27](#_Toc311402445)

[Host Selection in a Scenario 28](#_Toc311402446)

[Environment Selection on Test Hosts 30](#_Toc311402447)

[Installing DAF 32](#_Toc311402448)

[Pre-requisites 32](#_Toc311402449)

[Installing the DAF server 32](#_Toc311402450)

[Installing the DAF agent 32](#_Toc311402451)

[Running DAF 33](#_Toc311402452)

[Running the DAF server 33](#_Toc311402453)

[Running the DAF agent 33](#_Toc311402454)

[Uninstalling DAF 35](#_Toc311402455)

[Uninstalling the DAF server 35](#_Toc311402456)

[Uninstalling the DAF agent 35](#_Toc311402457)

# 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
* 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.

DAF server

SQL database

Webserver

System Under Test  
under test

DAF Agent

Test Stand

System Under Test  
under test

DAF Agent

Test Stand

System Under Test  
under test

DAF Agent

Test Stand

System Under Test  
under test

DAF Agent

Test Stand

DAF Server Machine Stand

Figure - Multiple Test Stand DAF Configuration

System Under Test

Test Host 1

DAF Agent

Test Host 2

DAF Agent

Linux  
Testhost N

DAF Agent

Linux Server

DAF server

Other components   
switch

Test Equipment Etc

SQL database

Webserver

IP network

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

# 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 host | x86 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.

# Licence Format

The DAF licence is a 20 byte object with the following fields:

Iu16 Type;  
Iu32 Serial;   
Iu32 Origin;  
Iu32 Duration;  
Iu16 Steps;  
Iu16 Spare;  
Iu16 Checksum;

|  |  |  |
| --- | --- | --- |
| Type | Indicates the type of the licence. The only valid type defined is 10. | Iu16 |
| Serial | Indicates the serial number of the licence. Serials start at 567. A serial in the 0-566 range is invalid. | Iu32 |
| Origin | The origin date of the licence, as a unix timestamp | Iu32 |
| Duration | The duration in days of the licence, if set to 0 the licence is valid forever | Iu32 |
| Steps | The max number of concurrent steps that may be scheduled, if set to 0 then any number is allowed | Iu16 |
| Spare | Not used. Must be 0. | Iu16 |
| Checksum | 16 bit XOR over the other fields so that the total 16 bit XOR is 0 for the full 20 bytes. | Iu16 |

DAF licences are communicated to users as a 40 character hexadecimal representation of the 20 byte structure above.