Fusion TR-069 Testing Framework

User Manual

Version 1.0.1

Table of Contents

1 Document Introduction

1.1 Document Purpose

The purpose of this document is to serve as a user manual for the Fusion TR-069 Testing Framework: explaining what it is and how to use it.

1.2 Document Audience

The TR-069 Testing Framework is intended to be used with Fusion Shell by hardware and software engineers for the purpose of testing TR-069 compliance and uncovering bugs in hardware and software. Thus some basic knowledge of Fusion Shell is preferred, but a determined user with some technical experience (such as experience of working from a UNIX shell) should easily be able to follow the step-by-step instructions provided.

1.3 Document History

Version	Editor	Date	Changes
1.0	E. Jönsson	2012-07-05	Initial public version.
1.0.1	E. Jönsson	2012-07-11	Chapter on hardware setup added.
1.0.2	E. Jönsson	2013-04-10	Updates with command changes.

1.4 Acronyms and Abbreviations

Acronym	Explanation
ACS	Auto Configuration Server.
APS	Automatic Provisioning System.
Fusion	Pingcom's eXtensible APS with advanced features such as Service Windows, Job Control and Smart Groups.
CPE	Customer Premises Equipment. Used in this document to refer to a single physical device. Same as the term “Device”.
Parameter	Each individual configuration setting is represented in the Fusion Data Model as a Parameter. A Parameter consists of a name and usually (but not always) a value.
Unit	A dataset in the Fusion database consisting of Parameter values relating to a single CPE. This dataset may extend beyond the Parameter values actually sent to the CPE, as some Parameter values may only be useful or needed by the Fusion itself. Also, the dataset may represent only a subset of all the configurable settings in the CPE. For these reasons, it is important to distinguish the term “Unit” from the terms “CPE” and “Device”.
Profile	Dataset stored in the Fusion containing Parameter values shared by multiple Units of the same Unit Type. A Unit is always assigned to a single profile. Multiple Profiles may be created for a Unit Type.
Unit Type	Units that represent CPEs of the same model share a common definition of that CPE model named Unit Type. The Unit Type definition is a list of Parameter names only, as the Unit Type

	never contains any Parameter values (values are stored in the Unit and/or Profile).
Group	A set of matching criteria used to search for Units. Commonly referred to as Smart Group.
TR-069	Industry standard provisioning protocol used by the Fusion to read and write configurations from and to the CPEs, in addition to handle upgrades.
LAN	Local Area Network. Typically a network interface only reachable from the local network inside a customer premise.
WAN	Wide Area Network. Typically a network interface reachable from the Internet.

1.5 References

Document	
[1]	Fusion Production Description
[2]	Fusion Shell User Manual

2 Introduction

The Fusion TR-069 Testing Framework is a set of command line tools, scripts and utilities designed to run automated TR-069 compliance tests on different pieces of hardware. Conceptually, tests are relatively straightforward. The Fusion TR-069 Server could, for example;

- Attempt to retrieve a number of parameters from a provisioned Unit, and make sure the replied adhere to the TR-069 standard.
- Attempt to set a number of parameters and then reading them back, making sure they have been correctly stored. (Persisting through reboots)

While this document will attempt to explain basic usage of commonly used commands, it is out of scope to describe every command in detail. For such, please see [2].

3 Logging in

Ideally, you have been given shell access to a Fusion provisioning server, such as xaps-a.owera.com. In that case, log in from a UNIX shell like this:

```
$ ssh <username>@xaps-a.owera.com
```

Provide your password when prompted. After having established a secure session to the server, you will once again be asked for your login credentials. Simply do so, and you will be greeted by Fusion Shell.

4 Important Fusion Shell concepts

At this point, it is worth giving a reminder about some commonly used Fusion Shell commands, along with important concepts such as the shell "context".

The context is similar to the concept of "working directory" from operating systems such as Windows and UNIX. Many UNIX shell commands use the working directory as an implicit parameter – The 'pwd' command prints it, the 'cd' command changes it, file paths are given relative to it, and so forth.

Fusion Shell uses its context in the same manner. Many commands are only valid in certain contexts. These contexts are:

- Unit Type. (ut:) A set of parameters common to all Units of a single device model.
- Profile. (pr:) A Unit Type can contain many different profiles, or *values* for a subset of the parameters in its parent Unit Type.
- Unit. (un:) A profile can contain many different Units, a representation of a single device.

It goes without saying that when issuing Fusion Shell commands, it is important to be issuing them in the right context. Section 4.1 through 6 gives a brief overview of the most commonly used Fusion Shell commands. In a pinch, one can list the most up-to-date documentation by using the 'help' command, such as:

```
help setparam
```

For a full reference, please see [2].

4.1 Changing contexts

The two most commonly used Fusion Shell commands to change context are 'cc' and 'Unit'.

The most basic command is 'cc', change context. 'cc' works much like 'cd' in a UNIX shell, it can be used to navigate up and down the context hierarchy. Some examples:

Changing context to the "root" context, meaning every Unit Type:

```
cc /
```

Changing context to the 'HydrogenHA' Unit Type:

```
cc ut:HydrogenHA
```

Changing context to the Unit '000000-HydrogenHA-000001' found in the 'default' profile, of the 'HydrogenHA' Unit Type:

```
cc ut:HydrogenHA/pr:default/un:000000-HydrogenHA-000001
```

To easier change context to a specified Unit, use the 'Unit' command. It is essentially a limited version of 'cc' that switches only to Unit contexts, but contains a search filter. For example:

```
unit Hydrogen-000001
```

Provided that the match is unique, the context will be changed to the specified Unit, regardless of what profile or Unit Type it has.

4.2 Working with parameters

Two operations are essential when working with parameters: we need to be able to list them, and to set them.

The commands for listing parameters are differently named in the different contexts. In the Unit Type and Profile contexts, it is called 'listparams' and takes an optional filter parameter. For example, to list all parameters (making sure the context is either Unit Type or Profile) related to testing, run:

```
listparams Test
```

In the Unit context, this command is instead named 'listUnitparams' and 'listallparams' and differs only slightly in output.

In all contexts however, there is only one command to set a parameter: 'setparam'. For example, to set the parameter 'System.X_OWERA-COM.TR069Test.Enable' to 1, we type:

```
setparam System.X_OWERA-COM.TR069Test.Enable 1
```

Again, remember to execute these commands in the appropriate context.

5 Establishing Unit connectivity

The first order of business is to make sure the Unit is easily accessible by the Fusion server so that TR-069 messages can be exchanged. There are two main methods of accomplishing this:

1. Adjusting the service window so that the Unit "phones home" more often. This is not a commonly used method, since it is both slow and has the disadvantage that the required parameter changes does not persists through reboots on some Units.
2. Set up a direct link so that Fusion can "kick" the Unit into action. This option has two variations:
    1. If the Unit is accessible on a public IP address, one can use kicks over HTTP/S and TCP.
    2. If the Unit is behind a gateway and not reachable over TCP, one can instead use UDP kicks with the help of the STUN protocol.

There is no need at this point to read up on the finer details of TCP or the STUN protocols, it is often enough to decide whether the Unit is behind a gateway or not.

5.1 Hardware setup

In order to perform provisioning, the unit (obviously) needs an internet connection. It's also recommended that this internet connection is piped through a hub or a managed switch with mirroring, so that all traffic on every port is mirrored to every other port. This way, by hooking up a computer onto the hub or managed switch, we can monitor all traffic passing through the testing device by using (for example) Wireshark. The network topology is pictured in 1.

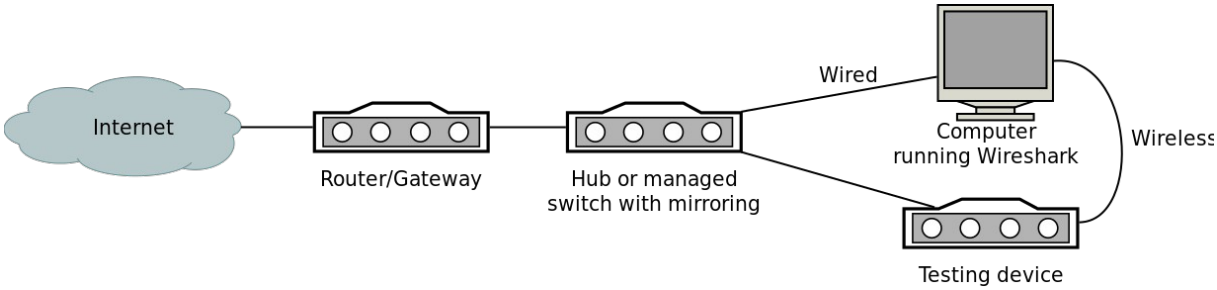

Illustration 1: Recommended hardware setup

Note that sometimes, it might be necessary to access the web interface of the test unit, and so we recommend a wireless connection between the testing device and the monitoring computer as well.

5.2 Adjusting the service window

The easiest, but least reliable method to set up Fusion connectivity is to adjust the service window. Most Units are configured to contact the provisioning server about once per day, which unfortunately is far too seldom when performing tests. To make it call home about once every minute instead, set the following Unit parameters:

```
setparam System.X_OWERA-COM.ServiceWindow.Spread    0
setparam System.X_OWERA-COM.ServiceWindow.Frequency 10000
```

5.3 Kick over a public IP

If the Unit is on a public IP, (that is, not behind a gateway) the Unit will notice this and publish a 'ConnectionRequestURL' parameter[1]. To force the Unit to contact the server and upload this parameter, simply restart it or issue a 'refresh' command. To test the connection, issue a 'kick' command:

```
kick
STUN server will be instructed to initiate a ConnectionRequest, will wait max 30
sec. to see outcome.
STUN server has changed the provisioning mode back to PERIODIC
The CPE should have been notified of the kick - if nothing happens in 30 sec ask
customer to reboot device
SUCCESS: Device has connected to the TR-069 server and updated Fusion
```

Afterwards, check the 'LastConnectTms' parameter from the Unit context: it should be updated to a recent timestamp:

```
listunitparams LastConnect
"Unit Type Parameter Name"        "Unit Parameter Value/Provisioned Value"
System.X_OWERA-COM.LastConnectTms "2012-07-05 09:43:37"
```

[1] If both 'ConnectionRequestURL' and 'UDPConnectionRequestAddress' are set, Fusion will by default use the method described in section 5.4 instead.

5.4 Kicking behind a gateway

If the Unit is behind a gateway, (or indeed, if only one method is ever to be used) the recommended method is to set up STUN connectivity. The parameters needed can be listed from the Unit context with:

```
listallparams STUN
```

As with the ConnectionRequestURL parameter, the Unit will publish a 'UDPConnectionRequestAddress' parameter which will primarily be used.

As described in section 5.3, make sure that this parameter has been uploaded to the Fusion server by restarting the Unit or issuing a 'refresh' command. We test connectivity in the same manner: by issuing the 'kick' command and checking the 'LastConnectTms' parameter from the Unit context.

6 Working with test cases

Test cases are essentially tiny programs, or scripts, that lists one or more instructions to perform, and states what the expected result is. Each Unit Type has its own set of test cases. To list these, use the 'listtc' command from the Unit Type context. Like many other list-commands in Fusion Shell, 'listtc' takes optional filter parameters. For example, to list all test cases that tests values (as opposed to attributes) among all read-only parameters, run:

```
listtc VALUE NULL [READONLY]
```

The last argument in square brackets belong to a class of filter parameters called “tags”. A tag is simply an arbitrary string generated when the test case was created, used in order to improve filtering capabilities.

Test cases can be inspected. As evident from the output from the 'listtc' command, test cases also have a unique identifier, an ID. Pass this ID as argument to the 'showtc' command, and all data related to that particular test case till be printed:

```
showtc 1205
```

6.1 The 'testsetup' command

Once a suitable set of test cases has been found, (by using the filter capabilities of 'listtc' as described in the previous section) we are ready to set up a Unit for testing. This is done by running the 'testsetup' command from the Unit context.

Now, a test case can contain a different mix of “getting” parameters, “setting” parameters, rebooting and resetting the Unit, and so forth. It often makes sense to run all of these in a specified order, such as all the SET methods first, followed by all the GET methods etc. To specify in what order we would like our test steps to proceed, we can pass a ordered, comma-separated list to the option named 'steps'. If we don't, it simply defaults to GET, i.e. only performing the GET step. Furthermore, we might sometimes want to reset our device to the factory default settings before running a test. If so, we specify 'reset=true'. By default, this option is false.

There is also the 'tag-filter' option, which (much like 'listtc') filters which tests we want to run. Continuing with our running example, we could run:

```
testsetup steps=GET tag-filter=[READONLY]
```

This would translate into “For all test cases matching the [READONLY] tag, run the GET step.”

Another example would be:

```
testsetup steps=SET,REBOOT,GET reset=true tag-filter=[READWRITE]
```

This would mean “Reset the test unit, then, for all test cases matching our search, execute all the SET commands first, followed by a reboot, followed by all the GET commands.”

As always, the 'help' command is useful to see all capabilities of the 'testsetup' command:

```
help testsetup
```

6.2 Executing tests

When we are happy with our test setup, it is time to run 'enabletest'. This command is executed from the Unit context, takes no parameters, and puts the Unit in “Testing Mode”:

```
enabletest
```

Being in Testing Mode means that the next time the Unit would contact the Fusion TR-069 server, testing will commence. This can be forced by issuing a 'kick' command from the Unit context:

```
kick
```

Alternatively, one can reboot the Unit or simply wait until the next service window comes around. Finally, if the tests should be aborted for any reason, (for

7 Evaluating tests

After having established connectivity in one way or the other, (section 5) selected and executed a test run, (section 6) it is time to evaluate the test runs. Depending on what tests are running, the Unit might become temporarily unavailable, in particular if the tests involve restarting and/or resetting the Unit. Once the Unit is back up however, the test history can be inspected by running 'listtesthistory' from the Unit context:

```
listtesthistory
```

The output from the command will vary, depending on what tests have been performed. If any of the tests have failed, a (hopefully) useful error message will have been printed to help with troubleshooting.

You can also filter the output from 'listtesthistory' using the options 'startTms', 'endTms', 'result' and 'failed' among others. An example would be:

```
listtesthistory failed=true expectError=false result=%Time%
```

See 'help listtesthistory' for a full description of the options available.