Test ACS Data Sheet

Overview

The Gatespace Test ACS is designed to automate the testing of Customer Premise Equipment (CPE) for compliance with the TR-069standard. One or more test scripts can be scheduled for a specific device. Those scripts are then executed when the device makes an INFORM contact with the test ACS. Test scripts are self modifying in that they can evaluate the value of parameters as reported by the device being tested to determine the next branch to execute within the script. A given test may be performed over multiple INFORMS thereby allowing device functions that consume significant time or a device reboot to be tested.

Any organization with responsibilities for testing devices for TR-069 compliance can achieve significant productivity improvements using the Gatespace Test ACS. Scarce resources can be devoted to the design of the testing regimen and freed from the tedious and unproductive task of manually setting up, executing and evaluating a large numbers of largely repetitive tests.

Typical organizations using the Test ACS are device manufacturers and service providers with multiple functional areas each having a slightly different perspective on product testing/evaluation:

- Product development test
- Product integration test
- Release test
- Conformance (TR-069) test
- Performance test
- Specialized testing of deployed devices

While each functional area may put different emphasis on the features being tested, most tests can be drawn from a common pool of reusable test scripts.

Test Scripts

In the Test ACS test scripts are described as **Service Definitions** that contain one or more **Service Operations**. Individual devices are subscribed to one or more Service Operations. When the device connects to the Test ACS to make an INFORM the service operations to which the device has been subscribed are executed serially. The services have access to device parameter values plus any variables carried forward from earlier executions of the script for the specific device.

During execution the Service Operation can evaluate device parameter values, interact with the device under test, write information to the test log and conditionally transfer control to another Service Operation within the Service Definition. Test scripts, service operations, are expressed as XML files that are easily imported and exported. A test ACS can use any combination of standard scripts provided by Gatespace, custom scripts developed by Gatespace (or third parties), and scripts developed in-house by the customer.

Test Volume

Testing for TR-069 compliance can be a long and laborious process with several hundred basic tests before even thinking about handling imperfectly constructed ACS requests and complex interactions between multiple ACS requests. The test process is further exacerbated by the need to repeat the battery of tests following any failure.

The Gatespace Test ACS addresses the problem of test volume by allowing multiple related tests to be packaged together so that all the tests are performed by scheduling the single package or script. Once a test script is scheduled and the subject device is configured with the address of the test ACS, testing is fully automated. The tester can leave the device and the Test ACS to work through a potentially very large number of tests. Executing a full set of tests towards the end of a development cycle may require an extended period. The tester simply schedules the tests and reviews the resulting test log at the end of the cycle.

TR-069 Compliance

The end objective of most Test ACS users is to achieve verifiable compliance with TR-069. As with many standards of this type, what constitutes compliance is not comprehensively defined and the number of test cases that can be imagined is almost limitless. By using the Test ACS the testing organization can show (verify) that a specific device responded in the expected (correct) manner to a battery of tests. Generally a test script (Service Operation) will exercise a particular aspect of TR-069 with a correct, or well-formed, messages and (more importantly) a series of incorrect, or badly-formed, messages simulating an out of compliance ACS.

Host System Requirements

The Test ACS runs on a typical workstation. The recommended minimum configuration is:

- 1Ghz processor
- 1 Gb memory
- 500 Gb disk
- Red Hat ES-4 or ES-5 with default workstation suite

Capacity

The capacity, in terms of devices supported, size of database, and transaction rate of the full license Test ACS is not 200 concurrently registered devices. Most Test ACS deployments never encounter capacity or performance limitations. Typically a test organization has <100 devices registered and will have <10 devices in active test at any time. Testing is not particularly sensitive response times in either direction during script execution and tests are not usually subject to human monitoring in real time.

Building Test Scripts

The following shows a Service Definition with several Service Operations for testing port forwarding to a LAN device. Note that the first Service Operation, id=”create’ sets is automatically executed, following subscription to a device, to set up the environment that allows the other Service Operations to execute.

```
<ServiceDefinition name="IGDPortForwarding" version="1.0"
long-name="Port forward to a LAN Device"
provider="Gatespace Networks, Inc." object-model="tr069"
object-model-version="1.0">
<ServiceObjects clone="true">
        <!-- SBVar without values must specify a
                source attribute. The only source currently
                defined is "dialog"             -->
        <SBVar name="protocol" value="TCP" />
        <SBVar name="client-ip" source="dialog" />
        <SBVar name="external-port" source="dialog" />
        <SBVar name="internal-port" source="dialog" />
        <SBVar name="description" source="dialog" />
        <!--  Dialog defines the source for the above
                SBVar variables.                -->
```

```
        <Dialog title="Port Forwarding Parameters">
                <Input name="client-ip" label="LAN Server IP" input="field"
                        width="18" />
                <Input name="external-port" label="External TCP Port"
                        defvalue="80" input="field" width="7" />
                <Input name="internal-port" label="Internal Port"
                        defvalue="80" input="field" width="8" />
                <Input name="description" label="Description"
                        defvalue="Port Mapping Desc" input="field" width="30" />

        </Dialog>
        <!-- This ServiceObject has an instance of "exact" which
                indicates that no interaction with the CPE is necessary
                to determine the parameter path names.
                DEFAULTWANCONNECTION is a built-in function that returns
                the object path name of the default WAN connection.
        -->

        <ServiceObject name="default-wan-connection"
                base="${DEFAULTWANCONNECTION}" instance="exact" />
        <ServiceObject name="port-mapping"
                base="${default-wan-connection}PortMapping."
                instance="create-if-needed">
                <!-- all parameters must match. If any miss-match a new instance if created -->
                <Match key="InternalClient" value="${client-ip}" />
                <Match key="InternalPort" value="${internal-port}" />
                <Match key="ExternalPort" value="${external-port}" />
                <Match key="PortMappingProtocol" value="TCP" />
        </ServiceObject>
</ServiceObjects>

<ServiceOps>
        <ServiceOp id="create" name="Port Forward Service">
                <State name="run-1">
                        <SetParameterValues log="Set Port forward Parameters">
                                <ServiceParameter
                                        name="${port-mapping}PortMappingEnabled" value="1" />
                                <ServiceParameter
                                        name="${port-mapping}PortMappingLeaseDuration"
                                        value="0" />
                                <ServiceParameter name="${port-mapping}RemoteHost"
                                        value="" />
                                <ServiceParameter name="${port-mapping}ExternalPort"
                                        value="${external-port}" />
                                <ServiceParameter name="${port-mapping}InternalPort"
                                        value="${internal-port}" />
                                <ServiceParameter
                                        name="${port-mapping}PortMappingProtocol"
                                        value="${protocol}" />
                                <ServiceParameter
                                        name="${port-mapping}InternalClient" value="${client-ip}" />
                                <ServiceParameter
                                        name="${port-mapping}PortMappingDescription"
                                        value="${description}" />

                        </SetParameterValues>
                        <Responses>
                                <!--Expected Responses -->
                                <SetParameterValuesResponse
                                        log="Port Forwarding parameters have been set"
                                        state="DONE"
                                        result="ok" resultMsg="Port Forward Service Set" />
                                <Fault
                                        log="Invalid arguments ${SetParameterValuesFault}"
                                        state="DONE"
                                        result="failed" resultMsg="Failed" />
                        </Responses>
                </State>
        </ServiceOp>
        <ServiceOp id="disable" name="Disable Port Forward Service">
                <State name="dis-1">
                        <SetParameterValues log="Set Port forward Disable">
```

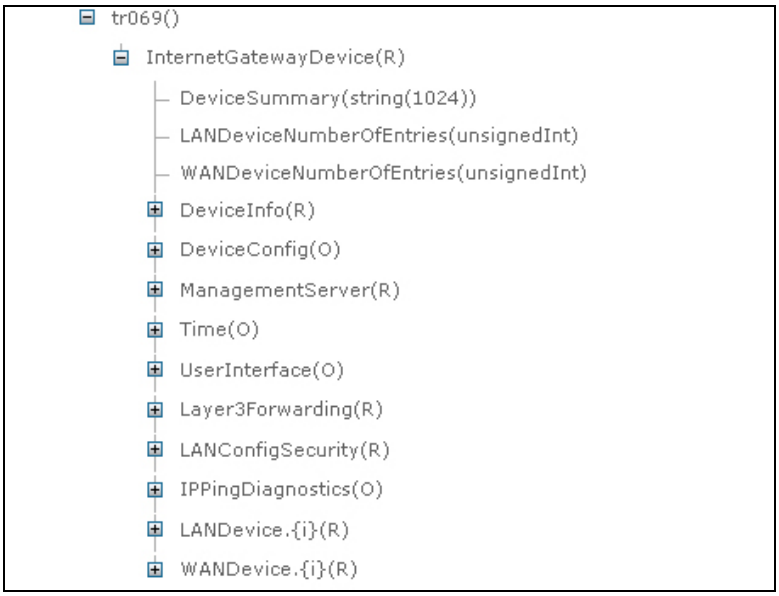
```
                                        <ServiceParameter
                                                name="${port-mapping}PortMappingEnabled" value="0" />
                                </SetParameterValues>
                                <Responses>
                                        <!--Expected Responses -->
                                        <SetParameterValuesResponse
                                                log="Port Forwarding Disable has been set" state="DONE"
                                                result="ok" resultMsg="Port Forward Disable" />
                                        <Fault
                                                log="Invalid arguments ${SetParameterValuesFault}"
                                                state="DONE"
                                                result="failed" resultMsg="Failed" />
                                </Responses>
                        </State>
                </ServiceOp>


                <ServiceOp id="delete" name="Remove Port Forwarding">
                        <State name="del-1">
                                <DeleteObject ObjectName="${port-mapping}" />
                                <Responses>
                                        <DeleteObjectResponse
                                                log="Port forwarding object deleted" state="DONE"
                                                result="ok"
                                                resultMsg="Port Forward Deleted" />
                                        <Fault log="DeleteObject fault" state="DONE"
                                                result="failed" resultMsg="Failed" />
                                </Responses>
                        </State>
                </ServiceOp>

                <!-- There are no other service actions required for this service -->
        </ServiceOps>
</ServiceDefinition>
```

Data Models

The Test ACS supports multiple data models. Typically, a device manufacturer will define a new data model for each combination of vendor specific extensions to the standard TR-069 data model. The standard TR-069 data model currently extends to >520 lines. The standard model, exploded to the second level, is shown below:



Acquiring the Gatespace Test ACS

To acquire the Gatespace Test ACS please visit www.gatespace.com