dBm069 Testing Service
Description of the Test Cases executed

(Example)

dBm Ingenieros S.L.
C/ Ayala 75

28006 Madrid (ESPAÑA)

Tel. +34 638 743 807

Email: sales@dBmIngenieros.com
Web: www.dBm069.com
Updated: Sept. 4th 2018

0

0 Introduction

dBm069 is a software that automates TR-069 testing. In order to perform an automatic Test Plan, dBm069 makes use of a library, which is an XML file that contains the definition of the test cases. With dBm069 you can test a device independently of its data model, as long as you have the right library.

The aim of this document is to provide a detailed description of the test cases contained in the test report that has been generated by dBm069. This information is contained in Chapter 3. Chapters 1 and 2 contain additional information to give you information about how dBm069 executes the Test cases.

Chapter 1 includes the description of the general structure of a test case, as well as the different tools that dB069 offers the users to create their own test cases.

Chapter 2 includes a list of all the different actions (complementary commands) that dBm069 uses to be able to perform the Test cases automatically.

Description of all test cases included in the test report is included in Chapter 3.

1

1 General structure of automatic test cases

An automatic Test Case is a text written in XML format that describes all the steps that dBm069 must perform to execute a TR-069 test. This text can be understood by a human and also by the execution engine of dBm069 (once the Library is encrypted, that is, converted from .xml to .lib format).

Information in automatic test cases is organized in three different blocks: Sessions; Sequences; and Requirements.

1.1 Sessions

A session has the same meaning as a TR-069 session. It contains RPC methods, and it can contain more than one RPC method if wished. When the ACS establishes a session, CPE must respond with an Inform that contains the EventCode “6 ConnectionRequest”. Every time that the user wants to implement a RPC method in the library, she needs to put it inside a session.

1.2 Sequences

A sequence is a section of the library that does not contain RPC methods, but complementary commands that dBm069 uses to complete a test plan. These complementary commands inside sequences are called actions, and dBm069 uses them for different purposes: to detect an Inform with a particular EventCode, to upload a FW from the PC to the server to set the environment for a test where it is necessary to download the file through TR-069, to open the http server, etc. Every time that the user wants to implement an action in the library, she needs to put it inside a sequence.

In all the dBm069 actions the parameter “Report” can be configured. This parameter contains two subparameters:

1. SuccesMessage: This parameter allows defining the message that will be shown at the html report generated by dBm069 when the result of the executed action is success.

2. ErrorMessage: This parameter allows defining the message that will be shown at the html report generated by dBm069 when the result of the executed action is failure.

Note that it is important to be careful defining these messages because sometimes the expected result of an action is that it fails, so a success of the action means a failure of the test case.

1.3 Requirements

Requirements contain all the conditions that dBm069 needs to check to decide whether a test has passed or not. Not all test cases need the definition of requirements. Requirements are only needed in those cases where in order to decide if the test passes or not a verification of one or some conditions is needed.

Requirements contain two elements: Condition and Fail Message. The fail message will be shown when the result of the evaluation expressed in the condition is False. Logical expressions are used to check the conditions.

A condition is an expression that returns a logical value (true or false). Expressions can contain variables, constant values, operators, parenthesis, and functions. Expressions are fully described in next section.

Requirements will be executed only if all the previous steps of a test (sessions and sequences) have been executed without causing an execution error. If any session or sequence has caused an execution error, dBm069 will show an execution error message, and the test will finish without checking the Requirements.

If there are no execution errors, and all conditions defined at the Requirements section are correct, then the test result will be “Pass”.

2

2 Different kind of Actions

As it was explained before, actions are complementary commands that can be used inside sequences, that dBm069 uses for different purposes, such us to detect an Inform with a particular EventCode, to open the http server, etc.

This section describes the different actions that can be used in a sequence and its parameters.

2.1 PatternMatch

There are some parameters that belong to a specific multi-instance object, and in order to be able to execute automatic tests where those parameters are used, it is important to know the number of this instance. The value of the instances of the object that are defined in a CPE can change between different equipment, as well as on the history of tests performed in a specific sample, so when performing manual tests the user needs to read manually the data model of the equipment in order to find out which is the number of the instance that contains the parameter that needs to be modified. When performing automatic tests with dBm069, those instances can be read automatically using the action “PatternMatch”. This allows preventing manual intervention during tests execution.

PatternMatch is thus an action that dBm069 uses to find out the instance of the object that will be used as the parameter or part of it in any part of the test case.

PatternMatch action needs to use some RPC methods to read the name of the objects and their instances that are going to be used in a test case. dBm069 software can use to different RPC methods in order to obtain these objects:

1. GPN method: PatternMatch uses this method by default.

2. GPV method: PatternMatch uses this method only if the “RPC” parameter is included in this action. This method is included in this action because we have found some CPEs that do not support the GPN RPC method.

2.2 WaitInform

WaitInform is an action that dBm069 uses to detect an Inform (containing a specific EventCode, or simply the first to arrive).

2.3 WaitTransferComplete

WaitTransferComplete is an action that dBm069 uses to detect the Inform that contains the EventCode “TransferComplete” after the transfer of a file.

2.4 IPConfig

IPConfig is an action that dBm069 uses to read the data of a network connection of the PC (similar to the console IPconfig command). This is the way that dBm069 uses to check that the real data of a network connection are the same as the data configured and read by means of TR-069.

2.5 LanPorts

LanPorts is an action that dBm069 uses to enable or disable either all the Ethernet network adapters connected to a specific IP, or one specific Ethernet network adapter. This is the way that dBm069 uses to connect and disconnect automatically the complementary equipment necessary for some test cases, e.g. GTW.TR069.098.f.hstlan.i.

All the Ethernet network adapters that are disabled using this action are enabled again at the end of the test by this action.

When this action is used in a test case, it is necessary to define an element of the following two depending on whether the user wants to disable or to enable the network adapters:

1. Disable: allows the user to disable the Ethernet network adapters connected to a specific IP.

Note that before disabling an Ethernet network adapter it is necessary to execute the action “IPconfig release” in order to release the IP that is using this network adapter.

2. Enable: allows the user to enable all the Ethernet network adapters that have been previously disabled.

2.6 WlanNetworks

WlanNetworks is an action that dBm069 uses to detect the list of available WLAN networks, i.e. the list of available Wi-Fi networks.

2.7 WlanTestConnection

WlanTestConnection is an action that dBm069 uses to connect the PC to a specified WLAN network, i.e. to a specified Wi-Fi network. Once that it has been possible to check the connection to the specified WLAN, this action disconnects it.

2.8 WlanNavigate

WlanNavigate is an action that dBm069 uses to navigate on different websites using the WLAN connection, i.e. using the Wi-Fi connection. This action performs the following operations:

1. First the PC is connected to the specified Wi-Fi network.

2. The PC navigates on the Internet addressing to the specified web site.

3. Finally, the WLAN connection is disconnected.

2.9 Navigate

Navigate is an action that dBm069 uses to navigate in different websites using the LAN connection of the CPE side.

2.10 NetworkShare

NetworkShare is an action that dBm069 uses to access a shared directory . This is the way that dBm069 uses to check the access and the contents of a USB memory stick connected to the CPE.

2.11 Http Server

dBm069 contains a http server that enables to perform automatically all the tests that include upload and download of files using a http server.

Http server is created/started and stopped during execution of the test plan using actions.

Next follows the description of all actions that can be performed with the http server.

2.11.1 HttpServer/Start

HttpServer/Start is an action that dBm069 uses to create/open the http server if it has never been created before, or to start it, if it has been previously created.

2.11.2 UploadFile

UploadFile is an action that dBm069 uses to upload a file from the PC to the http server. This is the way that dBm069 uses to put in the http server the files that will be needed for executing the tests. Note that before executing this action, the server must be previously started.

2.11.3 DownloadFile

DownloadFile is an action that dBm069 uses to download a file from the http server to the PC. This action is used when the objective is to overload the line at the down direction. Note that before executing this action, the server must be previously started.

2.11.4 HttpServer/GetFileList

HttpServer/GetFileList is an action that dBm069 uses to get the list of files that are located in the http server or to check if a particular file is located in the http server.This functionality is not defined in http protocol, but this action has been implemented to allow this verification in test cases that require it.

2.11.5 HttpServer/Delete

HttpServer/Delete is an action that dBm069 uses to delete either a specific file or the whole list of files that are located in the http server. The server must be previously started. This is not a proper function of the http protocol but this action has been implemented in our http server because it is necessary to check the operation of some tests.

2.11.6 HttpServer/Stop

HttpServer/Stop is an action that dBm069 uses to stop the http server.

2.12 Ftp Server

dBm069 contains a Ftp server that enables to perform automatically all the tests that include upload and download of files using a ftp server.

Ftp server enables upload and download of files using ftp. Server is created when required and it uses the proper ports of a ftp connection.

Next follows the description of all actions that can be performed with the ftp server.

2.12.1 FtpServer/Start

FtpServer/Start is an action that dBm069 uses to create/open the ftp server if it has never been created before, or to start it, if it has been previously created.

2.12.2 UploadFile

UploadFile is an action that dBm069 uses to upload a file from the PC to the ftp server. This is the way that dBm069 uses to put in the ftp server the files that will be needed for executing the tests. Note that before executing this action, the server must be previously started.

2.12.3 DownloadFile

DownloadFile is an action that dBm069 uses to download a file from the ftp server to PC. This action is used when the objective is to overload the line at the down direction. Note that before executing this action, the server must be previously started.

2.12.4 FtpServer/GetFileList

FtpServer/GetFileList is an action that dBm069 uses to obtain the list of files that are located in the ftp server or to check if a particular file is located in the ftp server. This funcition is not defined in ftp protocol, but this action has been implemented to allow this verification in those test cases which require it.

2.12.5 FtpServer/Delete

FtpServer/Delete is an action that dBm069 uses to delete either a specific file or the whole list of files that are located in the ftp server. This is not a proper function of the ftp protocol but this action has been implemented in our ftp server because it is necessary to check the operation of some tests.

2.12.6 FtpServer/Stop

FtpServer/Stop is an action that dBm069 uses to stop the ftp server.

This action does not have any parameter because it can only be used if there is an ftp server started, and the action will stop the ftp server that was previously opened in the test.

2.13 TlsCipherTest

TlsCipherTest is an action that dBm069 uses to check if the https server of the CPE uses any of the encryptions included in a list, in one particular or different communication protocols.

2.14 TestOpenPort (Traffic Generator/Analyzer)

dBm069 contains a traffic generator/analyzer that enables sending UDP or TCP traffic to any port range with a specified speed (bitrate). This is the way that dBm069 uses to check if a port or a port range is open or closed and to check if a filter is working correctly.

dBm069 uses the action “TestOpenPort” to make some operations with the traffic generator/analyzer in a test.

TestOpenPort is an action that dBm069 uses to check if a port or a port range is really open or closed after it has been open/closed by means of TR069. dBm069 checks if a port

range is open sending some traffic through this port range and detecting if this traffic is received.

The process that dBm069 uses to check the traffic is described bellow:

1. Traffic is generated from the support router to the public IP address of the CPE using the external port range specified.

2. This traffic should be received in the CPE side using the internal port range specified.

3. Test duration is 1 minute.

4. Action result:

    a. If 25% of the expected traffic is received at every port, the action result will be “success” and it will finish, because this will show that the port is really open.

    b. If 25% of the expected traffic is not received at any port after a minute, the action result will be “failure” and it will finish. In this case the list of internal ports that do not receive the expected traffic will appear at the “Events” window of dBm069.

dBm069 traffic generator/analyzer contains:

1. An agent that transmits the traffic and that is executed in the server (http server).

2. An agent that receives the traffic and that is executed at the PC where dBm069 is installed.

Both agents are controlled by the PC where dBm069 is installed.

dBm069 traffic generator/analyzer is very versatile and it enables the execution of tests using up to the layer 4. The traffic generator/analyzer contains a TrafficShaping module

that enables sending packets with a very accurate speed, although it is based in a PC and the jitter cannot be controlled it is enough for TR-069 testing.

2.15 Script

In order to be able to perform automatic tests, there are some tests that require the execution of scripts in order to prevent the loss of connection between the ACS and the CPE, as for example put the correct configuration file after doing a FactoryReset. When performing manual tests the user needs to change the configuration manually in each test where the change of the configuration can cause the loss of connection between the ACS and the CPE. When performing automatic tests, the change of the configuration file is executed automatically using a Script. This allows preventing that the software needs to stop during tests execution in order to allow the user to change the configuration of the CPE when the corresponding test is going to be executed.

Script is an action that dBm069 uses to make some modifications in the configuration of the CPE or to send some commands to the CPE using telnet, SSH or rs232.

3

3 Descriptions of Test cases

3.1 GTW.TR069.cp.au.1 (Authentication- Digest or Basic)

- **Description**

The goal of this test is to check that the CPE can successfully establish a CWMP session with the ACS using digest authentication.

- **Test Steps**

1. Open a sequence to detect the Inform that contains the eventcode “6 ConnectionRequest” using the action “WaitInform”. The value of the parameter Waittofinish must be false because we want to detect the Inform sent due to the next session. We also capture using an execution variable the authentication used by CPE.The parameter “Report” is used in order to define the success and error messages that are going to be shown at the html report.

2. Open a session to execute SetParameterValues method to disable the periodic Inform. It is necessary to disable the periodic Inform to avoid some confusions with the informs.

3. Requirements contain the following conditions.

   a. Conditions 1 and 2: Check that CPE uses digest authentication according to the standard (see TR-069 point 3.2.2.2). The OR operator is used to show the correct failure message when the authentication used by CPE is different than “Digest”.

3.2 GTW.TR069.cp.perinfo.1 (Periodic Inform - Operation)

- **Description**

The goal of this test is to check that the Periodic Inform functionality in the CPE is correct according to the standard requirements: it sends an Inform with the EventCode 2, and the measured periodic Inform interval is the same than the one configured in the equipment. This is checked for different values of the PeriodicInformInterval.

- **Test steps**

1. Open a session to execute the SetParameterValues method to enable the periodic Inform and to fix the periodic Inform interval.

2. Open a sequence to detect the Inform that contains the eventcode “2 Periodic” using the action “WaitInform”. The value of the parameter Waittofinish must be true because we want to wait to the finish of this action to continue with the test. We also capture using an execution variable when the periodic Inform is sent. The value of the parameter Timeout should be higher than the value of the periodic Inform to detect this Inform. The parameter “Report” is used in order to define the success and error messages that are going to be shown at the html report.

3. Open a session to execute GetParameterValues method to check that the values modified in the SetParameterValues method have changed correctly.

4. Open a sequence to detect the Inform that contains the eventcode “2 Periodic” using the action “WaitInform”. The value of the parameter Waittofinish must be true because we want to wait to the finish of this action to continue with the test. We also capture using an execution variable when the periodic Inform is sent and calculate the periodic Inform interval. The value of the parameter Timeout should be higher than the value of the periodic Inform to detect this Inform. The

parameter “Report” is used in order to define the success and error messages that are going to be shown at the html report.

5. Repeat the steps from 1 to 4 with a different value of the periodic Inform interval.

6. Repeat again the steps from 1 to 4 with a third different value of the periodic Inform interval.

1. Requirements contain the following conditions. Note that execution variables obtained from the SPV, GPV and waitInform are used to check conditions:

    a. Conditions 1 to 4: Check that the value of the parameters modified in the SetParameterValues methods have changed correctly

    b. Conditions 5 and 6: Check that CPE sends two informs that contain the eventcode (2) for the first value of the periodic Inform interval (First and second WaitInform must detect an inform)

    c. Conditions 7 and 8: Check that the first periodic Inform interval calculated using execution variable is the same than the first periodic Inform interval configured in the equipment (with a permissible deviation of 5 seconds)

    d. Conditions 9 and 10: Check that CPE sends two informs that contain the eventcode (2) for the second value of the periodic Inform interval (Third and fourth WaitInform must detect an inform)

    e. Conditions 11 and 12: Check that the second periodic Inform interval calculated using execution variable is the same than the second periodic Inform interval configured in the equipment (with a permissible deviation of 1 second)

    f. Conditions 13 and 14: Check that CPE sends two informs that contain the eventcode (2) for the third value of the periodic Inform interval (Fifth and sixth WaitInform must detect an inform)

g. Conditions 15 and 16: Check that the third periodic Inform interval calculated using execution variable is the same than the third periodic Inform interval configured in the equipment (with a permissible deviation of 1 minute)

3.3 GTW.TR069.cp.spv.2 (SPV - Multiple parameters)

- **Description**

The goal of this test is to check that it is possible to execute the SetParameterValues method in the CPE to configure the value of multiple parameters.

- **Test steps**

1. Open a session to execute SetParameterValues method to configure the value of two parameters.

2. Open a session to execute GetParameterValues method to check that the values modified in the SetParameterValues method have changed correctly.

3. Requirements contain the following conditions. Note that execution variables obtained from the SPV and GPV are used to check conditions:

    a. Conditions 1 and 2: Check that the values of the parameters modified in the SetParameterValues method have changed correctly.

3.4 GTW.TR069.cp.ao.3 (AO - Error Conditions)

- **Description**

The goal of this test is to check that the CPE returns the appropriate error code according to the standard if the AddObject is executed with:

1. A wrong parameter name.

2. An object name without a "." at the end of the name

3. A not multi-instance object

- **Test Steps**

1. Open a session to execute AddObject method to create a new instance of a multi-instance object that does not exist. The ExpectedResult of this method is set to Failure and the "IgnoreFail" element is included because, if TR-069 protocol is correctly implemented in the CPE, this method will fail and doing it this way, we will avoid this execution error allowing dBm069 being able to check the conditions defined at the Requirements block, using the execution variable =Error()(see "Multi-instance objects: Instance read at object creation" in order to understand how to use the instance)

2. Open a session to execute AddObject method to create a new instance of a multi-instance object without a "." at the end of its name. The ExpectedResult of this method is set to Failure and the "IgnoreFail" element is included because, if TR-069 protocol is correctly implemented in the CPE, this method will fail and doing it this way, we will avoid this execution error allowing dBm069 being able to check the conditions defined at the Requirements block, using the execution variable =Error()(see "Multi-instance objects: Instance read at object creation" in order to understand how to use the instance)

3. Open a session to execute AddObject method to create a new instance of a not multi-instance object. The ExpectedResult of this method will be ignore because, if TR-069 protocol is correctly implemented in the CPE, this method

will fail and it is necessary to avoid this execution error allowing dBm069 being able to check the conditions defined at the Requirements block, using the execution variable =Error()(see “Multi-instance objects: Instance read at object creation” in order to understand how to use the instance)

4. Requirements contain the following conditions. Note that execution variables obtained from the AO method are used to check conditions:

    a. Condition 1: Check that the FaultCode that read in the first AddObjectResponse method is greater than 0, i.e. the AO method is not executed correctly with a name of a parameter that does not exist.

    b. Condition 2: Check that the FaultCode read in the first AddObjectResponse method is the same as expected. The OR operator is used to avoid that this failure is shown when the AO method is executed correctly

    c. Condition 3: Check that the FaultCode read in the second AddObjectResponse method is greater than 0, i.e. the AO method is not executed correctly without a “.” at the end of its name.

    d. Condition 4: Check that the FaultCode read in the second AddObjectResponse method is the same as expected. The OR operator is used to avoid that this failure is shown when the AO method is executed correctly

    e. Condition 5: Check that the FaultCode read in the third AddObjectResponse method is greater than 0, i.e. the AO method is not executed correctly with a name of a not multi-instance object

    f. Condition 6: Check that the FaultCode read in the third AddObjectResponse method is the same as expected. The OR operator is used to avoid that this failure is shown when the AO method is executed correctly