DIMARK SOFTWARE, INC.

# Dimark Client Demo

v.4.0

## QUICK-START

February, 2011

A LASER FOCUS ON SECURE WAN MANAGEMENT

# Contents

# 1. Dimark Client Demo Overview

Included in this archive are Dimark TR-069 Clients compiled to run on a Linux desktop computer.

Two versions are provided, one which emulates a gateway (TR-098) device and one that emulates a LAN (TR-106) device, without having any features implemented in the background (Set/Get work, AddObject/ DeleteObject works).

Please note that while each client directory contains a data-model.xml file to define the respective data models, these files are not interchangeable (between TR-098 and TR-106 folders).

This client demo is based on the Dimark TR-069 Client source code that is distributed under license and deployed in a wide range of CPE Devices (DSL, VoIP, STB, WiMax, Femto, etc.).

# 2. About Dimark's Embedded TR-069 Client

The Dimark TR-069 client provides a complete TR-069 implementation, including, but not limited to TR-069, TR-098, TR-104, TR-106, TR-110, TR-111, TR-135, TR-140, TR-142, TR-143 and TR-196, as well as other third party specifications, such as WiMax Forum Specification for TR-069 devices.

The key functions offered by these protocols is as follows:

- TR-069 enables an extensible, secure, communications layer, while also providing basic gateway router and Wi-Fi configuration and management functionality [1].
- TR-098 enables QoS functionality and provides configuration profiles to ease management and deployment of gateway devices [2].
- TR-104 and TR-110 combine to provide remote VoIP device configuration and management [3, 4].
- TR-106 and TR-111 combine to allow the remote management of devices on a LAN, even those using the private IP space behind a NAT gateway [5, 6].
- TR-135 enables the configuration and management of Set Top Boxes (STB) [7].
- TR-140 enables the configuration and management of Network Attached Storage (NAS) [8].
- TR-142 enables the configuration and management of PON devices [9].
- TR-143 enables network throughput performance tests and statistical monitoring [10].
- TR-157 enables component objects for CWMP [11].
- TR-181 defines data models for Device and Internet Gateway Device [12].
- TR-196 enables support for Femtocell (cellular repeater) devices [13].

The Dimark TR-069 client is typically provided as ANSI C source code that runs on embedded Linux, with the API designed to ease the integration of the client into existing environments. Partners have also quickly deployed the client on non-Linux-based RTOS platforms (VxWorks, Nucleus, etc.), as well as WinCE.

The client incorporates an abstraction layer that will speed implementation as well as make the addition of many new features, upgrades and updates virtual drop-ins.

The client incorporates all updates that have resulted from more than five years of field use, feature requests and customer deployments.

# 3. Starting Dimark Client Demo

## Prerequisites Check

> Starting from Client v. 4.0 in order to run and compile Dimark Client user should check the presence and version of the `crypto`, `ssl` and `xml2` shared libraries.

Please perform the following steps before running or compiling Dimark Client:

1. Run `rpm -qa | grep openssl` command.
It checks the presence of both `ssl` and `crypto` shared libraries.

2. Run `rpm -qa | grep libxml2` command.
It checks the presence of `xml2` shared library.

If user hadn't checked the libraries before compilation he can perform the libraries check when he already has an executable file. To perform the check for `dimclient` file, run the following command:

```
ldd dimclient
```

You will see the results presented in the form of two columns. First column reflects requested library name and version and the second reflects its availability:

```
linux-gate.so.1 =>  (0xffffe000)
libpthread.so.0 => /lib/libpthread.so.0 (0xf775a000)
libcrypto.so.6 => not found <--------> Library is not available
libssl.so.6 => not found<--------> Library is not available
libxml2.so.2 => /usr/lib/libxml2.so.2 (0xf760c000)
libc.so.6 => /lib/libc.so.6 (0xf74a0000)
/lib/ld-linux.so.2 (0xf779c000)
libdl.so.2 => /lib/libdl.so.2 (0xf749b000)
libz.so.1 => /lib/libz.so.1 (0xf7487000)
libm.so.6 => /lib/libm.so.6 (0xf745d000)
```

If you see the errors stating that some needed libraries are not found, the following recommendation may help you. Try to create symbolic links for missing libraries substituting them for existing libraries. For example:

```
# cd /usr/lib
# ln -s libcrypto.so.0.9.8 libcrypto.so.6
# ln -s libssl.so.0.9.8 libssl.so.6
```

## Operation

1. Unzip the Dimark Client Demo archive:

```
# unzip Dimark_Client_Demo.zip
```

2. From a terminal, change to the directory that contains the client you wish to run. For example:

```
# cd /usr/local/Dimark_Client_Demo/TR-098
```

or

```
# cd /usr/local/Dimark_Client_Demo/TR-106
```

3. Set the client binary and shell script to executable:

```
# chmod +x dimclient
# chmod +x start.sh
# chmod +x conv-util
```

4. Modify the data-model.xml file to point to the correct ACS URL (...ManagementServer.URL) and username password (...ManagementServer.Username, ....ManagementServer.Password). The default values connect the client to Dimark's online ACS.

5. Start the client:

```
# ./start.sh
```

The start.sh script creates the file system and data model file in /tmp folder and executes the client. Make sure to always restart the client using the start.sh script to ensure all housekeeping functions normally associated with a CPE reboot are performed.

The client will then communicate with the Dimark ACS. You can leave it running in a terminal and the client will periodically contact the ACS.

## Demo Client and NAT Traversal

Connection requests are supported by the Demo Client.

To enable connection requests when the demo client is running on a LAN with NAT, manually set the ...ManagementServer.ConnectionRequestURL in the data-model.xml file to use your gateway's public IP address in the appropriate URL (http:<ipaddress>:8082/acscall), open port 8082 on your gateway and forward all 8082 traffic to the client machine.

If you are unable to open a port to the firewall, all settings will take place during the next periodic inform.

# 4. Dimark Client Integration

The following sections are extracted from the programmer's manual and describe how to integrate the client into a host system and create extensions to the client.

Note: These sections are provided to give you an understanding of how the Dimark Client Source Code is integrated into your platform; this information is not required to run the Dimark Client Demo. However, certain parts of this document, particularly Compiler Flags and Client Configuration File Elements tables, will provide guidance if you wish to modify parameters or extend the data model during your evaluation.

For compilation development versions of the libraries are required:
`libxml2-devel, openssl-devel`.

## Compiler Flags for the Client

When building your reference system, make sure that compiler flags are set as needed on the target system to create an appropriate simulation.

| Compiler Flag | Location | Usage |
|---|---|---|
| ACS_REGMAN | Makefile | Enable support for the REGMAN ACS. Do NOT enable this option unless the REGMAN ACS is used as it is not fully compliant with regular TR-069. |
| FILETRANSFER_STORE_STATUS | globals.h | Enables / disables saving information about file transfer. Option is useful if device is rebooted while the file transfer is not completed. In this case saved information is used to resume the transfer. |
| HANDLE_NUMBERS_OF_ENTRIES | globals.h | Define if the client should update the NumberOfEntries of an Object during an AddObject or DeleteObject. |
| HAVE_ATMF5_DIAGNOSTICS | globals.h | Enables / disables ATMF5 diagnostics. |
| HAVE_AUTONOMOUS_TRANSFER_COMPLETE | globals.h | Enables / disables AutonomousTransferComplete method. |
| HAVE_DIAGNOSTICS | globals.h | Enables / disables all diagnostics. |
| HAVE_DOWNLOAD_DIAGNOSTICS | globals.h | Enables / disables DownloadDiagnostics method. Does not concern Download method. |
| HAVE_FACTORY_RESET | globals.h | Enables / disables FactoryReset method. |
| HAVE_FILE | globals.h | Enables / disables following flags: HAVE_UDP_ECHO, HAVE_GET_QUEUED_TRANSFERS, HAVE_SCHEDULE_INFORM, HAVE_REQUEST_DOWNLOAD and TransferComplete method. |
| HAVE_FILE_DOWNLOAD | globals.h | Enables/disables both Download and DownloadDiagnostics methods. |

| Compiler Flag | Location | Usage |
|---|---|---|
| HAVE_FILE_UPLOAD | globals.h | Enables / disables both Upload and UploadDiagnostics methods. |
| HAVE_GET_ALL_QUEUED_ TRANSFERS | globals.h | Enables / disables GetAllQueuedTransfers method. If this flag is disabled AutonomousTransferComplete method (HAVE_AUTONOMOUS_TRANSFER_COMPLETE flag) is also automatically disabled. |
| HAVE_GET_QUEUED_ TRANSFERS | globals.h | Enables / disables GetAllQueuedTransfers method. |
| HAVE_HOST | globals.h | Enables / disables host interface support. |
| HAVE_IP_PING_ DIAGNOSTICS | globals.h | Enables / disables IP Ping diagnostics. |
| HAVE_KICKED | globals.h | Enables / disables Kicked method. |
| HAVE_OPTIONAL | globals.h | Enables / disables CPE optional responding methods: Download, Upload, FactoryReset, GetQueuedTransfers, GetAllQueuedTransfers, ScheduleInform, SetVouchers, GetOptions; and ACS optional calling methods: GetRPCMethods, RequestDownload, Kicked. |
| HAVE_REQUEST_DOWNLOAD | globals.h | Enables / disables RequestDownload method. |
| HAVE_RPC_METHODS | globals.h | Enables / disables GetRPCMethods method. If GetRPCMethods is enabled it will be called on initial connection of CPE. GetRPCMethods call returns list of methods that can be called on ACS by the means of CPE. If some optional ACS methods are not supported on ACS they will be automatically switched off on CPE. |
| HAVE_SCHEDULE_INFORM | globals.h | Enables / disables ScheduleInform method. |
| HAVE_UDP_ECHO | globals.h | Enables / disables UDP Echo diagnostics (TR-143 standard). |
| HAVE_UPLOAD_ DIAGNOSTICS | globals.h | Enables / disables UploadDiagnostics method. Does not concern Upload method. |
| HAVE_VOUCHERS_OPTIONS | globals.h | Enables / disables both SetVouchers and GetOptions methods.<br>Vouchers are set in the CPE by an ACS. And Vouchers' data structure instructs a particular CPE to enable or disable Options, and characteristics that determine under what conditions the Options persist. |
| HAVE_WANDSL_ DIAGNOSTICS | globals.h | Enables / disables WAN DSL diagnostics. |

| Compiler Flag | Location | Usage |
|---|---|---|
| NO_LOCAL_REBOOT_ON_CHANGE | globals.h | Disable automatic reboot when changes are made to parameters that normally require a reboot.<br>This flag enables / disables ignoring reboot flag in the data-model.xml file (if local flag is enabled in data-model.xml file the reboot must be done by the ACS on parameter change).<br>If NO_LOCAL_REBOOT_ON_CHANGE flag is enabled local flag is ignored and therefore a status return code of 1 is returned to signal that the parameter change has be done but not confirmed yet. |
| TR_111_DEVICE | Makefile | Defines that the client will operate as a TR-111 Device. This results in the client using an object model starting with “Device.” |
| TR_111_DEVICE_WITHOUT_GATEWAY | Makefile | Same as TR_111_DEVICE, however at startup the client will not wait for data for the Gateway to be filled before contacting the ACS. |
| TR_111_GATEWAY | Makefile | Defines that the client will operate as a TR-111 Gateway device. This results in the client using an object model starting with “InternetGatewayDevice.” |
| WITH_COOKIES | Makefile | This is a required flag for the Dimark Client. This flag should always be present. User has a possibility to compile Dimark client without cookies but the client will not be able to connect to Dimark ACS. |
| WITH_OPENSSL | Makefile | Required if you build with OpenSSL. This flag should always be present. |
| WITH_SOAPDEFS_H | globals.h | Enables / disables soapdefs.h file.<br>This flag should always be present. |
| WITH_SSLAUTH | Makefile | Enable SSL Authentication through certificates. |
| WITH_STUN_CLIENT | Makefile | Enable STUN ACS connections |
| WITHOUT_DEBUG=TRUE | Makefile | Defines that compilation of client debug information will be disabled and it will not be included to code. |

## Client File Structure

The client provides the option to modify the way it stores data. The default implementation that the client ships with is using a regular file system which might not be available on the target system.

The src/host directory contains all File I/O related access methods that store data, events, options, etc. This

allows updating the code without loosing your File I/O modifications.

The default implementation uses several directories to store files and are defined in src/host/storage.h

| Define | Typical Value | Usage |
|---|---|---|
| PERSISTENT_PARAMETER_DIR | ./tmp/parameter/ | Directory where the list of added parameters are stored |
| PERSISTENT_DATA_DIR | ./tmp/data/ | Directory where the parameters are stored |
| PERSISTENT_FILE | ./tmp/bootstrap.dat | Persistent client data that must survive a Reboot and a Boot, but not survive FactoryReset |
| EVENT_STORAGE_FILE | ./tmp/eventCode.dat | Event Storage of the client that must survive a reboot. |
| DEFAULT_DOWNLOAD_FILE | ./download.dwn | Default download file |
| PERSISTENT_OPTION_DIR | ./tmp/options/ | Directory where TR-069 Option data is stored |
| VOUCHER_FILE | ./tmp/Voucher_%d | Voucher creation file string (files must survive reboot) |

The location of those files should be changed to be in line with the file systems used on the CPE.

Note: The start.sh script creates and manages the data in the file storage during use of the demo.

## Integration Process

The integration process consists of four steps:

- Expanding the TR-069 object model
- Implementing Init/Get/Set Methods
- Implementing Host interface
- Implementation of File System I/O (if required)

It is recommended to start by expanding the TR-069 Object model and then implementing the Set/Get functionality to retrieve and store parameter data from the host system. The final step should be adding the host interface to call the TR-069 client so that Notifications from the CPE to the ACS are possible.

### Expanding the TR-069 Object Model (data-model.xml)

The client reads the data-model.xml configuration file (path to xml file is customizable in start.sh) and will be

read on each startup of the client.

Based on cwmp-datamodel-1-2.xsd [14] Dimark generated extended data-model.xml file that is supported by Client. Developed xml schema allows to describe both basic datamodel and datamodel converted from dps.param (formerly used Dimark specific configuration file). To modify Broadband Forum xml schema (cwmp-datamodel-1-2.xsd) Dimark added new parameter attributes such as `notification`, `maxNotification`, `reboot`, `initIdx`, `getIdx` and `setIdx`.

If you have dps.param instead of xml configuration file, run an executable conv.jar file in console mode. It allows to convert dsp.param to xml file.

Sample command:

```
java -jar conv.jar dps.param data-model
```

where:

`dps.param` – parameter that allows to state name and path to xml file that will be created. If no path is stated (as in the sample) file will be created in the directory where conv.jar is located. Model described in xml will have the same name as xml file;

`data-model` – parameter that allows to state name and path to xml file that will be created. If no path is stated (as in the sample) file will be created in the directory where conv.jar is located. Model described in xml will have the same name as xml file.

### XML Configuration File Description

The name of *.xml file is presented in the following form:

```
<model name="data-model">
```

As the result you will get data-model.xml configuratin file.

data-model.xml that is shipped in one package with Dimark Client describes what Object Model the client is using. Provided data-model.xml file includes the general DeviceInfo and ManagementServer settings. These are read-only data that lists CPE parameters and provides URL to contact the ACS.

*1. TR-069 Parameter Name*

To state the fully qualified TR-069 name of the parameter in data-model.xml you should use the following structure:

```
<object base="Device." access="readOnly" minEntries="1"
 maxEntries="unbounded">
  <object base="Device.ManagementServer." access="readOnly"
   minEntries="1" maxEntries="unbounded">
    <parameter base="URL" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="1" initIdx="1"
     getIdx="1" setIdx="1">
    </parameter>
  </object>
</object>
```

where:

`Device.` – the top-level object for Device.ManagementServer TR-069 parameter;

`ManagementServer.` – the second-level object for Device.ManagementServer TR-069 parameter;

`URL` – the last element of Device.ManagementServer TR-069 parameter

`notification="0" maxNotification="2" reboot="1" initIdx="1" getIdx="1" setIdx="1"` – details that characterise the Device.ManagementServer TR-069 parameter (see their descriptions in the table below);

`access="readOnly" minEntries="1" maxEntries="unbounded"` – additional parameters that always should be included in every tag that defines object;

`access="readOnly" instance="0"` – additional parameters that always should be included in every tag that defines parameter.

*2. Data Type*

The data type of the element is represented by <syntax> tag in the following way:

```
<object ....>
    <parameter ... >
       <syntax>
         <string/>
       </syntax>
    </parameter>
</object>
```

where:

`string` – data type of the element.

There can be another types of the elements instead of `string`.

List of possible data types of the element (formerly presented as the second parameter in the dps.param syntax line):

- String (formerly 6 in the dps.param file);
- Integer (formerly 7 in the dps.param file);
- UnsignedInteger (formerly 9 in the dps.param file);
- Boolean (formerly 18 in the dps.param file);
- DateTime (formerly 11 in the dps.param file);
- Base64 (formerly 12 in the dps.param file);
- hexBinary;
- Int.

Types for DefaultObject (formerly 103, 104, 106, 115, 108, 109 in the dps.param file) are represented as corresponding data types for `default type="object"`.

*3. Access List*

Access List – comma separated list of access rights (i.e. Subscriber).

The following example of access rights has two entries – `x` and `y`:

```
<accessList>
  <entry>x</entry>
  <entry>y</entry>
</accessList>
```

If there is only one entry, the example will be as follows:

```
<accessList>
  <entry>x</entry>
</accessList>
```

*4. Default Value*

Default Value - the initial value of the parameter (only needed if Set/Get methods are not used).

Default Value is stated in the <syntax> tag below the data type tag and is presented as follows:

```
<syntax>
 <string/>
   <default type="factory" value="http://test.dimark.com:8080/dps/TR069"/>
</syntax>
```

*5. Details that Characterise the TR-069 Parameter*

An example from a TR-111 Device is:

```
<object base="Device." access="readOnly" minEntries="1"
maxEntries="unbounded">
   <object base="Device.ManagementServer." access="readOnly"
    minEntries="1" maxEntries="unbounded">
    <parameter base="URL" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="1" initIdx="1"
     getIdx="1" setIdx="1">
    </parameter>
   </object>
</object>
```

where:

`notification="0" maxNotification="2" reboot="1" initIdx="1" getIdx="1" setIdx="1"` – details that characterise the Device.ManagementServer TR-069 parameter.

The following table represents the definitions of Details that Characterise the TR-069 Parameter:

| Detail Element | Description |
|---|---|
| `notification` | Default setting for notification towards the ACS.<br>0 = no notification<br>1 = passive notification (during the next regular Inform)<br>2 = active notification (initiate an Inform when value changes)<br>3 = always include in Inform |
| `maxNotification` | The maximum permissible notification setting for this parameter (allows disabling of notifications through the ACS) |
| `reboot` | Flag to indicate if a Reboot is required if the value of the parameter is changed (from the ACS).<br>0 = Reboot<br>1 = No reboot required |
| `initIdx` | Allows calling an initialization function when the data is loaded from configuration file data-model.xml. Set to -1 if no initialization is required. |
| `getIdx` | The index number that was given to this parameter in the getArray[] of paramaccess.c<br>Set to -1 if the parameter does not have a Get method, and is Write Only (i.e. Passwords).<br>An entry of 0 reads the value from RAM<br>An entry of 1 reads the data from the persistent layer |

| Detail Element | Description |
|---|---|
| setIdx | The index number that was given to this parameter in the setArray[] of paramaccess.c.<br>An entry of -1 treats the parameter as Read Only<br>An entry of 0 causes the value to be read from the CPE<br>An entry of 1 reads the data from the persistent layer |

### Creating Multiple Objects

TR-069 allows definition of Objects that appear multiple times under a single parent object.

The following example illustrates how nested objects can be created.

```
<object base="Device.Services.STBService.1.Capabilities.VideoDecoder."
 access="readOnly" minEntries="1" maxEntries="unbounded">
    <parameter base="VideoStandards" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="0" initIdx="1" getIdx="1"
     setIdx="-1">
       <syntax>
         <string/>
         <default type="factory" value="MPEG2-Part2,MPEG4-Part4,MPEG4-
          Part10"/>
       </syntax>
     </parameter>
</object>
<object base="Device.Services.STBService.1.Capabilities.VideoDecoder.
 MPEG2Part2." access="readOnly" minEntries="1" maxEntries="unbounded">
    <parameter base="AudioStandards" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="0" initIdx="1" getIdx="1"
     setIdx="-1">
       <syntax>
         <string/>
         <default type="factory" value="MPEG2-Part3-Layer2,MPEG2-Part3-
          Layer3"/>
       </syntax>
    </parameter>
    <parameter base="ProfileLevelNumberOfEntries" access="readOnly"
     notification="0" maxNotification="2" instance="0" reboot="0"
```

```
      initIdx="1" getIdx="1" setIdx="-1">
        <syntax>
        <unsignedInt/>
          <default type="factory" value="1"/>
        </syntax>
      </parameter>
</object>
<object base="Device.Services.STBService.1.Capabilities.VideoDecoder.
 MPEG2Part2.ProfileLevel." access="readOnly" minEntries="1"
 maxEntries="unbounded"/>
<object base="Device.Services.STBService.1.Capabilities.VideoDecoder.
 MPEG2Part2.ProfileLevel.1." access="readOnly" minEntries="1"
 maxEntries="unbounded">
    <parameter base="Profile" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="0" initIdx="1" getIdx="1"
     setIdx="-1">
        <syntax>
          <string/>
          <default type="factory" value="HP"/>
        </syntax>
    </parameter>
    <parameter base="Level" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="0" initIdx="1" getIdx="1"
     setIdx="-1">
        <syntax>
          <string/>
          <default type="factory" value="H-14"/>
        </syntax>
    </parameter>
    <parameter base="MaximumDecodingCapability" access="readOnly"
     notification="0" maxNotification="2" instance="0" reboot="0"
     initIdx="1" getIdx="1" setIdx="-1">
        <syntax>
          <string/>
          <default type="factory" value="10240"/>
        </syntax>
    </parameter>
</object>
```

### Implementing a Set/Get Method

In order to implement an Init, Set or Get method it must be declared in the paramaccess.c module in the getArray[] or setArray[] respectively. These arrays contain a mapping from the index to the method name. There are many examples of access methods in the paramaccess.c file to use as a baseline for your own methods.

Once the method is implemented, the configuration file needs to be modified to include the Set/Get index number in the corresponding parameter entry.

Some of the index entries are already assigned by the client and cannot be changed.

- -1 No action;
- 0 Read/Write of the data is done without accessing the paramacess.c functionality;
- 1 Read/Write is done through paramaccess.c functionality.

Using entry 1 allows the client to handle the parameter in a read/write fashion without providing any backing implementation of the parameter. This way the client configuration file (data-model.xml) can be exploded to the full object model without completing the full integration work.

When defining index numbers, it is recommended to use the same number for both Set and Get methods to simplify debugging. Also, the client will use the Get index number during access from the host system when there is no Set access allowed from the access (ACS read-only parameter).

### Implementing the Host Interface

Since TR-069 requires host system to communicate with the TR-069 client it provides not only the ACS interface, but also a secondary interface that allows the host system to retrieve and set client information.

This secondary interface uses an HTTP port (defined in ./src/globals.h that defaults to port 8081) that is used to communicate with the host system. This listener is started as a separate thread inside the client so that communication is possible even while the client is already communicating with the ACS.

The distinction of this system is that it does not have the same restrictions as the ACS (access to Read-Only parameter data). It allows access to any data stored inside the client.

In order to retrieve information from the Dimark TR-069 client GET and PUT methods had been implemented.

In the following examples the notation <NL> (LF, 0x0a) means the single newline character and not CRLF (0x0d 0x0a) as on Windows.

### Retrieving Data from the Client

In order to read data from the client the following command would be issued over the host interface:

```
get<NL>
Parameterpath<NL>
```

The client would respond to this request with the following response:

```
Type<NL>
Parametervalue<NL>
Error code<NL>
```

This allows the CPE device to retrieve values that are stored inside the client environment. The code for this command is in hosthandler.c. The code calling the client must be done as part of the integration for parameters with notification support. This is done by issuing a HTTP GET from the host system towards the Dimark Client.

### Setting Client Data

This command is important for the implementation as it can trigger notifications (or alarms) towards the ACS. It does not store the value if it is managed outside the client through Set/Get methods.

The structure of the call is:

```
set<NL>
Parameterpath<NL>
Parametervalue<NL>
```

A notification is sent to the ACS if necessary. The parametervalue is restricted to 1024 chars.

### Adding Instance for an Object

In order to add new instance for an object the following command would be issued over the host interface:

```
add<NL>
Objectpath<NL>
```

The instancenumber of the new created object is returned, or if an error occurred a 0 is returned.

### Deleting an Instance Object

In order to delete an instance object this method can be used. The format is:

```
del<NL>
Objectpath<NL>
```

If no error occurred a OK is returned.

### Deleting an Option

This method can be used in order to delete an option which has timed out. The format is:

```
opt<NL>
VoucherSN<NL>
remove NL
```

Note: Some commands are available only for concrete data models if conditions (see table with available Compiler Flags in Compiler Flags for the Client section) are specified in code.

In this case vouchers options should be available:

```
#ifdef HAVE_VOUCHERS_OPTIONS
```

### Retrieving Vendor Data

In order to read vendor data from the client the following command would be issued over the host interface:

```
GETVENDORINFO
 send vg
```

The client would respond to this request with the following response:

```
 receive <errorcode> 0 = noError
 receive <manufactorerOUI>
 receive <serialNumber>
 receive <productClass>
 receive <emptyline>
```

### Setting Vendor Data

In order to set vendor data the following command would be issued over the host interface:

```
SETVENDORINFO
 send vs
 send <manufactorerOUI>
 send <serialNumber>
 send <productClass>
```

The client would respond to this request with the following response:

```
 receive <errorcode> 0 = noError
 receive <emptyline>
```

### Adding Device Data

In order to add data about connecting device the following command would be issued over the host interface:

```
ADDDEVICEINFO
 send da
 send <manufactorerOUI>
 send <serialNumber>
 send <productClass>
```

The client would respond to this request with the following response:

```
 receive <errorcode> 0 = noError
 receive <emptyline>
```

### Deleting Device Data

In order to delete device data the following command would be issued over the host interface:

```
REMOVEDEVICEINFO
 send dr
 send <manufactorerOUI>
 send <serialNumber>
 send <productClass>
```

The client would respond to this request with the following response:

```
 receive <errorcode> 0 = noError
 receive <emptyline>
```

### Retrieving ACS URL

In order to read ACS URL the following command would be issued over the host interface:

```
DHCPDISCOVERY
 send hd
 send <manufactorerOUI>
 send <serialNumber>
 send <productClass>
```

The client would respond to this request with the following response:

```
receive ho
receive <manufactorerOUI>
receive <serialNumber>
receive <productClass>
receive <URL>
receive <emptyline>
```

### Retrieving Gateway Data

In order to connect to Gateway device sends request with its own parameters and receives Gateway parameters to establish a connection. The format is:

```
DHCPREQUEST
 send gr
 send <manufactorerOUI>
 send <serialNumber>
 send <productClass>
```

The client would respond to this request with the following response:

```
receive ho
receive <manufactorerOUI>
receive <serialNumber>
receive <productClass>
```

## Additional Parameters

Additional parameters (specific for Dimark Client and not listed in any standards) were added to initial parameter file ().

`Dimark.` object contains all additional Dimark Client specific parameters.

Previously all transfers were allowed only from/to ACS. `TransferURL` parameter allows to designated location from which downloads/uploads will be started.

For Device.Dimark:

```
<object   base="Device.Dimark."   access="readOnly"   minEntries="1"
```

```
maxEntries="unbounded">
    <parameter base="TransferURL" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="1" initIdx="-1"
     getIdx="0" setIdx="-1">
       <syntax>
         <string />
         <default type="factory" value="[IP|URL]" />
       </syntax>
    </parameter>
</object>
```

where:

IP|URL is an IP address or URL of the site from which Download/Upload methods are allowed to be invoked besides ACS (for Download/Upload methods call format see section 15. Download and Upload Methods Support).

For InternetGatewayDevice.Dimark:

```
<object base="InternetGatewayDevice.Dimark." access="readOnly"
 minEntries="1" maxEntries="unbounded">
    <parameter base="TransferURL" access="readOnly" notification="0"
     maxNotification="2" instance="0" reboot="1" initIdx="-1"
     getIdx="0" setIdx="-1">
       <syntax>
         <string />
         <default type="factory" value="[IP|URL]" />
       </syntax>
    </parameter>
</object>
```

where:

IP|URL is an IP address or URL of the site from which Download/Upload methods are allowed to be invoked besides ACS (for Download/Upload methods call format see section 15. Download and Upload Methods Support).

## Implementing Parameter Storage

The storage interface used by the client has been completely redesigned in release 3.0 and improved in further

versions of the Dimark TR-069 Client.

In previous versions of the Dimark TR-069 Client, data was saved with both the current settings as well as Meta information (i.e. Access rights). Now those information sections are saved separately. The complete storage interface has been extracted into the file host/parameterStore.c. By separating those two elements, flash storage requirements are decreased since only persistent and changeable parameters are saved.

The supplied implementation counts on saving the data in two distinct directories. In each case the parameter name will be used as the filename as well.

The following functionality must be implemented in parameterStore.c. to read the initial parameter file (data-model.xml) line by line and use the callback function to perform additional work on each of the parameters and store the data:

```
int loadInitialParameterFile(newParam *callbackF)
```

In case the parameter defines an Init method it will call this Init method for the parameter.

The following function loads a single parameter file and create the parameter in memory:

```
int loadSingleParameterFile( const char *, newParam *callbackF )
```

The following adds a new parameter and saves the metadata (notification, access rights) of the parameter in persistent memory:

```
int storeParameter(const char *name, const char *data)
```

The metadata is given as part of the string and are identical to the entries in data-model.xml. The name is the complete fully qualified parameter name. The value of the parameter is not provided.

The following function is called when metadata of a parameter needs to be updated. The name field is the fully qualified name of the parameter:

```
int updateParameter(const char * name, const char *data)
```

The following function removes a parameter from persistent memory. This needs to remove both the metadata as well as the parameter value information. This function is called during Delete Object functionality:

```
int removeParameter(const char *name)
```

The following function removes all parameter information. This function is called during a FactoryReset() that is triggered by the ACS:

```
int removeAllParameters(void)
```

The following function saves the value of a parameter given as name in the first argument. Since the value is dependent of the type, the type is given as well with the parameters:

```
int storeParamValue(cont char *, ParameterType, ParameterValue *)
```

The following function retrieves the value of a parameter, which name is in first argument:

```
int retrieveParamValue(const char *, ParameterType, ParameterValue *)
```

The following function needs to be implemented for checking of a parameter data and/or metadata file existence:

```
unsigned int checkParameter(const char *)
```

The return value of this function can be one of the following:

- 0 File does not exist;
- 1 Data file exists;
- 2 Metadata file exists;
- 3 Both Data and Metadata files exist.

## Implementing Storage Environments

Storing of events, file transfers and options are done similar to the storage of parameters. In case the provided implementation needs to be changed, consult the host directory (host/eventStore.c., host/filetransferStore.c. and host/optionStore.c. files) for a sample implementation.

### Events

The following function creates the storage environment for events. This typically results in a single file being created where events can be stored in the file system. Events are defined in TR-069 (i.e. “0 BOOTSTRAP”, “8 DIAGNOSTICS COMPLETE”, “2 PERIODIC”, etc):

```
int createEventStorage(void)
```

The following function reads all events from the storage (created with createEventStorage) and calls newEvent() to process it in dimclient:

```
int readEvents(newEvent *func)
```

The following function removes all events from the event storage (created with createEventStorage):

```
int clearEventStorage(void)
```

The following function insert the event string (data) at the end of event storage (created with createEventStorage):

```
int insertEvent(const char *data)
```

The following bootstrap marker allows the client to detect if it's in an "initial" state (after factory reset or first time boot) or if this is a regular reboot situation. In the default implementation there is a marker inside the file. This marker must be retained over reboot events:

```
int createBootstrapMarker(void)
```

When the ACS issues a Factory Reset, the following method will be called to remove the Bootstrap marker from the event list and allow the client to appear with a BOOTSTRAP event against the ACS:

```
int deleteBootstrapMarker(void)
```

The following function is called during startup of the client to identify if the bootstrap marker is present. Returns true if bootstrapMarker is found else returns false:

```
bool isBoostrapMaker(void)
```

### File Transfers

The following function reads the list of open (pending) file transfers at the boot of the client. For each line the callback function has to be called:

```
int readFtInfor(newFtInfo *callbackF)
```

The following function saves the file transfer information with a specified name. The name is unique for each transfer:

```
int storeFtInfo(const char *name, const char *data)
```

The following function deletes the transfer data associated with a given name:

```
int deleteFtInfo(const char *name)
```

The following function deletes all informations about pending file transfers:

```
int clearAllFtInfo(void)
```

The following function returns a default filename as destination for a download in case the ACS did not deliver a destination filename in the download request:

```
const char * getDefaultDownloadFilename(void)
```

### Options

The following function is for processing vouchers, which are containers for options. The voucher must first be saved in a file. This file is used by gSOAP to read the XML information of the voucher:

```
const char *getVoucherFilename(int index)
```

The following function reads all options and initiates callback function for each dataset:

```
int reloadOptions(newOption callback())
```

The following function deletes all option data records:

```
int deleteAllOptions(void)
```

The following function deletes information for a named option:

```
int deleteOption(const char *name)
```

The following function saves data record under the provided name argument. The name field is unique:

```
int storeOption(const char *name, const char *data)
```

## Callbacks

Callbacks allow custom activities to be carried out in the client during specific situations. The defined callbacks are:

- Before the client makes a call to the ACS (preSessionCbList);
- After terminating the session with the ACS (postSessionCbList);
- Clean up when temporary storage is freed (cleanUpCbList).

The client needs to register the callback by calling the method addCallback, which is defined as:

```
void addCallback(Callback, List *);
```

The callback itself is defined as:

```
int (*Callback) (void);
```

No parameters are provided to those callbacks. The return value of the callback method itself can have the following values:

| Return | Description |
|---|---|
| CALLBACK_REPEAT | Will call the same callback the next time the trigger situation happens. |
| CALLBACK_STOP | Removes the callback from the list. The callback function will not be called again. |

Callback functions are called in the order they are registered.

## Diagnostics Support

Dimark Client supports following TR-143 diagnostics:

Download Diagnostics utilizing FTP transport and Download Diagnostics utilizing HTTP transport [15, 16].
The Download Diagnostics test is being used to test the streaming capabilities and responses of the CPE and the WAN connection.

Upload Diagnostics utilizing FTP transport and Upload Diagnostics utilizing HTTP transport [15, 16].
The Upload Diagnostics test is being used to test the streaming capabilities and responses of the CPE and the WAN connection.

UDP Echo is being used for monitoring. CPE acts as server and listens for UDP datagrams on UDP port 8088. When a datagram is received, the data from it is sent back in an answering datagram.

For further TR-143 diagnostics details please see TR-143, Enabling Network Throughput Performance Tests and Statistical Monitoring, Broadband Forum Technical Report.

You can browse skeleton function for IP Ping Diagnostics in `src/host/diagParameter.c`.

## Kicked RPC Support

Kicked RPC is supported by the Dimark Client v.3.0.10 and higher. Detailed description can be found in TR-069 CPE WAN Management Protocol v1.1 Amendment 2 (Annex A 4.2.1 and Annex D).

Sample of Kicked procedure invocation:
Type `http://<CPE_IP>:8084/command=com&arg=arg&next=NextURL` in your browser.
Where 8084 is port which CPE is listening to be kicked.
CPE has a DoS attaks protection mechanism.
Procedure invocation is case sensitive. Be sure you specify all parameters as they are shown in sample.

If Kicked RPC has been initiated successfully, browser is redirected to
`http://<CPE_IP>:8084/NextURL` page, defined in ACS KickedResponse RPC.

Otherwise you are redirected to `http://<CPE_IP>:8084/FaultURL` page or get HTTP 503 Error that mean that DoS attaks protection was not passed or site URL does not coincide with Device.ManagementServer. KickURL or InternetGatewayDevice.ManagementServer. KickURL.

## Download and Upload Methods Support

Download and Upload methods are supported by the Dimark Client. Dimark Client can transfer specified file from/to designated location. Detailed description can be found in TR-069 CPE WAN Management Protocol v1.1 Amendment 2 (Annex A 3.2.8 and Annex A 4.1.5).

Sample of Download method invocation:

```
http://192.168.15.17:8096/command=Download&type=1 Firmware Upgrade Image&url=http://www.insart.com/images/stories/content/homepage_logo.jpg&username=user&password=name&delay=10000&size=24024&target=target&success=success&failure=failure
```

Sample of Upload method invocation:

```
http://192.168.15.17:8096/command=Upload&type=2 Vendor Log File&url=http://www.insart.ua/post&username=user&password=name&delay=10
```

Here 8096 is port which CPE is listening to start downloads/uploads.

CPE has a DoS attacks protection mechanism.

Browser will generate corresponding success or failure message:

- HTTP 503 Error – transfer was not initiated as either DoS attacks protection was not passed or `command` parameter is incorrect or not specified or `url` parameter is not specified;
- transfer not accepted – transfer initiation was failed;
- transfer accepted – transfer was initiated (but that doesn't mean it have been started as delay time can be set).

Note: Only `command` and `url` are required parameters. If values of optional parameters are unknown they shouldn't be mentioned in methods invocation.
Methods invocation is case sensitive. Be sure you specify all method parameters as they are shown in samples. Sequence order of required parameters doesn't matter and their position can be shifted.

Sample of Download method invocation (required parameters):

```
http://192.168.15.17:8096/command=Download&url=http://www.insart.com
```

## Log Configuration

Dimark Client log can be configurated using log.config file located in root directory.

The following levels of logging exist in Dimark Client:

- DEBUG – debug data is logged (loggs everything). DEBUG logging level is implemented for environments where no debugging functionality is available. This option allows generating output of the client to help diagnose issues. Debug functionality is defined in debug.h and implemented in debug.c;
- INFO – information is logged;
- WARN – warnings are logged;
- ERROR – logs errors concerning stated subsystem.

Note: DEBUG level is the highest and loggs all data including INFO, WARN and ERROR. Each subsequent level include all logging levels that are below it.
If there are no log.config file, the default levels of subsystem logging will be INFO.

Debugs can be removed completely from the output and the program code. This is managed through compiler flag `WITHOUT_DEBUG=TRUE` in Makefile.

Each subsystem has assigned logging level.

The syntax of log.config file is as follows:

```
subsystem;level
```

The following sample includes all available subsystems and shows the example of log.config syntax:

```
SOAP;DEBUG <NL>
MAIN;INFO <NL>
PARAMETER;WARN <NL>
TRANSFER;ERROR <NL>
STUN;DEBUG <NL>
```

```
ACCESS;DEBUG <NL>
MEMORY;DEBUG <NL>
EVENTCODE;DEBUG <NL>
SCHEDULE;DEBUG <NL>
ACS;DEBUG <NL>
VOUCHERS;DEBUG <NL>
OPTIONS;DEBUG <NL>
DIAGNOSTIC;DEBUG <NL>
VOIP;DEBUG <NL>
KICK;DEBUG <NL>
CALLBACK;DEBUG <NL>
HOST;DEBUG <NL>
REQUEST;DEBUG <NL>
DEBUG;DEBUG <NL>
AUTH;DEBUG <NL>
<EOF>
```

where:

<NL> – new line symbol;

<EOF> – end of line symbol.

## Client Error Codes

The following table describes the Dimark specific error codes that can be generated. These error codes are specific to the Dimark Client and extend the TR-069 Error code table. Developers who will extend the Client and include additional error codes must make sure that they are using different error codes.

| Code | Constant | Description |
|---|---|---|
| 9800 | ERR_DIM_EVENT_WRITE | Unable to open persistent file for writing. This results in the client being unable to communicate status after reboot events. |
| 9801 | ERR_DIM_EVENT_READ | Unable to open persistent file for reading. This results in the client being unable to communicate status information to the ACS. |
| 9810 | ERR_DIM_TRANSFERLIST_WRITE | Unable to write a file transfer entry to persistent storage. This will result in the client being unable to process a DownloadRequest. |

| Code | Constant | Description |
|---|---|---|
| 9811 | ERR_DIM_TRANSFERLIST_READ | Unable to read file transfer entries from persistent storage. This results in the client being unable to process a DownloadRequest and its status to the ACS. |
| 9820 | ERR_INVALID_OPTION_NAME | The provided option does not exist in the CPE. This fault is only generated on the HOST interface and not used with the ACS. |
| 9821 | ERR_CANT_DELETE_OPTION | Unable to Delete a Voucher Entry This fault is only generated towards the HOST interface and not used with the ACS. |
| 9822 | ERR_READ_OPTION | Unable to read voucher information from persistent storage. |
| 9903 | ERR_WRITE_OPTION | Unable to write voucher information to persistent storage. |

# References

1. TR-069, CPE WAN Management Protocol (CWMP), Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-069_Amendment-2.pdf.
2. TR-098, Internet Gateway Device Data Model for TR-069, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-098_Amendment-2.pdf.
3. TR-104, DSLHomeTM Provisioning Parameters for VoIP CPE, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-104.pdf.
4. TR-110, DSLHomeTMApplying TR-069 to Remote Management of Home Networking Devices, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-110v1.01.pdf.
5. TR-106, Data Model Template for TR-069-Enabled Devices, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-106_Amendment-4.pdf.
6. TR-111, DSLHomeTMApplying TR-069 to Remote Management of Home Networking Devices, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-111.pdf.
7. TR-135, Data Model for TR-069 Enabled STB, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-135.pdf.
8. TR-140, TR-069 Data Model for Storage Service Enabled Devices, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-140_Amendment-1.pdf.
9. TR-142, Framework for TR-069 enabled PON Devices, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-142_Issue-2.pdf.
10. TR-143, Enabling Network Throughput Performance Tests and Statistical Monitoring, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-143_Corrigendum-1.pdf.
11. TR-157, Component Objects for CWMP, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-157_Amendment-3.pdf.
12. TR-181, Device Data Model for TR-069, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-181_Issue-1.pdf, http://www.broadband-forum.org/technical/download/TR-181_Issue-2.pdf.
13. TR-196, Femto Access Point Service Data Model, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-196.pdf.
14. cwmp-datamodel-1-2.xsd, Broadband Forum XML Schema http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-2.xsd
15. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt
16. RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt

# Runtime Questions

*Q. The client log shows ACS Notification failed message whenever there is connection request from ACS. Here is log for a connection request from ACS for TraceRouteDiagnostics. The Diagnostics runs properly and reports results back to ACS.*

```
<==================== LOG ======================>
soap_serve endet 0 env:Envelope :
GetAccess: 1 Device.ManagementServer.PeriodicInformEnable 0x11fe50
GetAccess: 1 Device.ManagementServer.PeriodicInformInterval 0x11e4c8
ACS Accept ret: 13
GetAccess: 119 Device.ManagementServer.ConnectionRequestURL 0x11f000
GetAccess:    1    Device.ManagementServer.ConnectionRequestUsername
0x11ee80
ACS Notification failed: 401      <--------> Error Message
ACS Accept ret: 13
Digest:    username=”dps”
Key: username Value: “dps
Digest:     realm=”Dimark”
Key: realm Value: “Dimark
Digest: nonce=”1234567890ABCDEF”
Key: nonce Value: “1234567890ABCDEF
Digest:    uri=”/acscall”
Key: uri Value: “/acscall
```

A. This is normal behavior as the ACS will not include the Authentication data in the first transmission. This allows the client to send the DIGEST information to the ACS which in turn will resend the request with the DIGEST Authentication information. This is normal HTTP behavior.

*Q. When running the client the following messages appear on the console:*

```
plugin/httpda.c(192): free(0x1002dcb8) pointer not malloced
plugin/httpda.c(192): free(0x1002b378) pointer not malloced
plugin/httpda.c(192): free(0x1002f250) pointer not malloced
plugin/httpda.c(192): free(0x1002e7d8) pointer not malloced
plugin/httpda.c(192): free(0x1002e2e0) pointer not malloced
plugin/httpda.c(192): free(0x1002f760) pointer not malloced
```

A. This happens when the client is compiled with the SOAP_DEBUG flag. Unfortunately the gSOAP DIGEST authentication component does not fully adhere to the coding standard of gSOAP and as a result, this message is displayed. It does not indicate a memory leak.

*Q: Why Basic authentication request is sent for Download regardless authorization type defined by ACS?*

A: Download process for Dimark Client v 3.0.10 and higher has the following peculiarity – regardless using authorization type, when download is initiated Basic authorization is sent by default. If server rejects initiating download with Basic authorization, Digest authorization is sent.
Such behavior doesn't contradict HTTP protocol standards. Problems can occur only if server doesn't correspond to HTTP 1.0 standard.