DIMARK SOFTWARE, INC.

Dimark Embedded Client

v.4.3

TECHNICAL REFERENCE & IMPLEMENTATION GUIDE

August, 2012

A LASER FOCUS ON SECURE WAN MANAGEMENT

Contents

Legal Notices

Copyright Notice

Warranty Disclaimer

THE DIMARK TECHNOLOGY AND ANY ACCOMPANYING MATERIALS AND INFORMATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND DIMARK EXPLICITLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

License

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. DIMARK SOFTWARE, INC. ("DIMARK") SOFTWARE IS LICENSED NOT SOLD.

FOR WARRANTY INFORMATION PERTAINING TO THIS PRODUCT, PLEASE REFER TO THE WARRANTY DISCLAIMER ABOVE.

1. License. The application, demonstration, system and other software accompanying this License, whether on disk, in read-only memory, or on any other media (the "Dimark Software"), and the related documentation are licensed to you by Dimark. You own the medium on which the Dimark Software are recorded, but Dimark and /or Dimark's licensor(s) retain title to the Dimark Software and related documentation. The License allows you to use the Dimark Software on a single Dimark product and make one copy of the Dimark Software in machine-readable form only for backup purposes. You must reproduce, on such copy, the Dimark copyright notice and any other proprietary legends that were on the original copy of the Dimark Software. You may also transfer all your license rights in the Dimark Software, the backup copy of the Dimark Software, the related documentation, and a copy of this License to another party provided the other party reads and agrees to accept the terms and conditions of this License.

2. Restrictions. The Dimark Software contains copyrighted material, trade secrets, and other proprietary material. In order to protect them, and except as permitted by applicable legislation, you may not decompile, reverse engineer, disassemble, or otherwise reduce the Dimark Software to a human-perceivable form: copy, modify, network, rent, lease, loan, or distribute the Dimark Software; or create derivative works based upon the Dimark Software in whole or in part. You may not electronically transmit the Dimark Software from one computer to another or over the network.

3. Termination. This License is effective until terminated. You may terminate the License at any time by destroying the Dimark Software, related documentation and all copies thereof. This License will terminate immediately without notice from Dimark if you fail to comply with any provision of this License. Upon termination you must destroy the Dimark Software, related documentation, and all copies thereof. Upon termination you shall remain subject to the provisions, restrictions and exclusions in this License Agreement and shall have no right to any refund of any amount paid for the Dimark Software. No termination shall release you from liability for any breach of this License Agreement.

4. Export Law Assurances. You agree and certify that neither the Dimark Software nor any other technical data received from Dimark, nor the direct product thereof, will be shipped, transferred, or exported, directly or indirectly, to any county in violation of any applicable law, including the United States Export Administration Act and the regulations there under.

5. Controlling Law and Severability. This License shall be governed by and construed in accordance with the laws of the State of California without regard to its conflict of laws provisions. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to affect the intent of the parties, and the remainder of this License shall continue in full force and effect.

6. Acknowledgment. You acknowledge that you have read this License Agreement, understand it, and agree to be bound by its terms and conditions. You also agree that the License agreement is the complete and exclusive statement of agreement between the parties and supersedes all proposals or prior agreements, oral or written, and any other communications between the parties relating to the subject matter of the License Agreement. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of Dimark.

1. Dimark TR-069 Client Integration

Version 4.3

This document describes how to integrate the client into a build system and performing extensions to the client. It also describes how to integrate the application into the host system so that notification events can dynamically be forwarded to the Auto configuration server (ACS).

The base client provided by Dimark provides a configuration file that defines some generic TR-069 parameters that identify the device and how to communicate with the ACS. All other functionality needs to be provided during the integration/implementation phase. This allows the client to operate with virtually any host system available.

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, TR-196 and TR-181, as well as other third party specifications, such as WiMax Forum Specification for TR-069 devices.

The key functions offered by these protocols are 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 eight years of field use, feature requests and customer deployments.

3. New in Dimark Client v4.3 Release

Version 4.3 contains a number of new features:

- TR-181 Configuration flag that allows to build client with TR-181 support.
- New memory manager was implemented. It facilitates more simple and evident control over .assignable and released memory.
- New data types were added: hexBinary, unsignedLong.
- Numbers of data types and objects were changed.
- The algorithm of UDPEcho Diagnostic functioning was improved. The full UDPEcho Plus support was added.
- Description of Uploads, Schedule Downloads was added to README section.
- New flag `with-tr181` was added into configuration. It allows to form makefile in which TR-181 support will be included. If `with-tr181` configuration flag is set, it results in the client using TR-181 data model.
- TR-181 data-model.xml file was added to project directory.
- Configuration flag `with-tr111` was changed to `with-device_root`.
- HTTPLogFilePathName and TESTLogFilePathName were added to dimclient.conf file.
- MaxDebugLogSize parameter was renamed maxHTTPLogSize, MaxDebugLogBackupCount was renamed maxHTTPLogBackupCount.
- -l command line parameter was added. It defines Log configuration file path (default “log.config”).
- DEBUG.log was renamed HTTP.log.
- The work with SQLite DB was accelerated in several places.
- The mutex “mem_usage_mutex” was deleted.
- Host interface of Dimclient was changed and is “telnet” compatible now.
- Targets were updated in the makefile.in.
- Support of SOAP Envelope Header with increasing cwmp:ID was implemented.
- Pretty-format of HTTP packets in the HTTP.log was implemented.
- Handling of Transfers, Deployment Unit and Schedule Inform was separated into the different threads.
- “Migration from v.4.2 to 4.3” section was added.

Version 4.3 contains the following major fixes:

- Handling of boolean and base64 data types was fixed.
- Potential Segmentation Faults were fixed.
- Not binding to any(0.0.0.0) IP interface if no -d argument passed on start.sh was fixed.
- Specific mutex for TransferList, DeploymentUnitList, ScheduleInformList was added.
- Segmentation fault that occurs when paramValue size > 1K was fixed.
- Single-quota character that was not allowed in string parameter value was fixed.
- HTTP Connection close that was not handled properly by dimclient was fixed.

- gSoap indefinite behavior when getter returns NULL value was fixed.
- Not proper uploading work was fixed. Handling of HTTP response after PUT message is sent was added.

Version 4.3 contains the following minor fixes:

- Usage of function sessionEnd() in the source code was fixed.
- RetrieveParamValue() function returns error code if parameter isn't found in DB.
- getCRU(), traverseParameter(), loadParameters() functions were fixed.
- http_post(), soap_getcookies() functions were fixed.
- Potential memory leak was fixed: soap_malloc() function was replaced with MemoryManger function.
- When 401 error comes from ACS for the second time in the session, the dimclient terminates the session.
- Function setParameter2Host() does not return OK (0) if error occurs.
- periodicIntervalDelay calculation in StunHandler was fixed.
- CR is not ignored when HTTP 400 Bad Request from ACS is received.
- debug.c respects all values from log.config.
- The problem with event codes deletion when "0 BOOTSTRAP" is in the eventCodeList was fixed.
- Minor editorial changes.

This section contains only major and noticeable changes. If you see other changes that are not reflected in documentation, you are welcome to contact Dimark at support@dimark.com to get detailed explanation about what was amended in that particular domain.

4. Migration from v.4.2 to 4.3

1. Download gSoap v.2.8.8 and execute apply_gSoap_patch.sh file to patch it (compulsorily).
2. Most files of the project were changed because of new features realization and bug fixing. Main changes are the consequences of:

- new memory manager realization;
- implementation of two new parameters types (hexBinary & unsignedLong);
- implementation of TR-181 data-model support.

For successful migration from v.4.2 to v.4.3 stick to the following instructions:

- Define all changes implemented by you to the v.4.2 client source code and move them to the v.4.3 client.
- Make sure that your code components do not conflict with new memory manager while moving the code. If necessary correct your code.
- If the handling of various parameter types took place in your code, maybe it is necessary to add handling of hexBinary and unsignedLong parameter types.

3. Pay attention to some renamed configuration and compilation flags (see “Compiler Flags for the Client” section of this document).
4. Pay attention that new -l command line attribute was added to the Dimclient application. This attribute serves to define Log configuration file path.
5. Pay attention to changes implemented to dimclient.conf file.
6. If you implemented [transfer]CallBack functions, pay attention to the changes of their interface in 4.3 version. If necessary correct your code.
7. Less significant features and fixes are stated in ChangeLog.v4.3.txt file in the source code of the project and “New in this Release” section of this document.

If you have problems with migration form v.4.2 to 4.3 or any questions, you are welcome to contact Dimark at support@dimark.com.

5. Client File Structure

In the Client file structure files are grouped according to purpose. Parts of a Client directory tree are listed below. All directories are grouped under the root directory named after current Client version.

- gsoap – stores files related with gSoap functionality.
  - soapdefs.h – using this file user can amend gSoap buffers size.
- patch – stores original gSoap files, that should be patched by apply_gSoap_patch.sh.
  - httpda.[ch].diff
  - md5evp.[ch].diff
  - smdevp.[ch].diff
  - stdsoap2.[ch].diff
  - struct_timeval.[ch].diff
  - threads.h.diff
- plugin – stores files responsible for digest access authentication (Upload, Download and ACS connectivity). Corresponding files will be borrowed from gSoap after user obtains commercial license agreement for gSOAP Version 2.8.8.
- src
  - ftp – stores files responsible for FTP Downloads and FTP Uploads.
    - cmds.[ch]
    - ftp.c
    - ftp_ft.[ch]
    - ftp_var.h
  - handlers – handlers of corresponding entities.
    - CRhandler.[ch]
    - DBhandler.[ch]
    - diagnosticsHandler.[ch]
    - hosthandler.[ch]
    - kickedhandler.[ch]
    - notificationhandler.[ch]
    - stunhandler.[ch] – flow that manages STUN client and UDP Connection Request listener.
    - timehandler.[ch]
    - transfershandler.[ch]
    - udpCRhandler.[ch]
  - host – stores files that allow to implement basic procedures (parameters and events storage, storage of information needed for Client data recovery in case of its Reboot, etc.).
    - diagParameter.[ch]
    - du_entryStore.[ch]
    - ethParameter.[ch]
    - eventStore.[ch]
    - filetransferStore.[ch]

    - optionStore.[ch]
    - parameterStore.[ch]
    - si_entryStore.[ch]
    - storage.h
  - options – stores files for SetVouchers and GetOptions methods implementation.
    - option.[ch]
  - profiles
    - downloadDiagnostics_profile.[ch]
    - ipping_profile.[ch]
    - time_profile.[ch]
    - traceroute_profile.[ch]
    - uploadDiagnostics_profile.[ch]
  - tr104 – stores files for operating with voice-over-IP (VoIP) devices.
    - voipParameter.[ch]
  - tr111 – stores files for operating with STUN protocol.
    - STUN_dimark.[ch] – UDP Connection Request listener.
  - callback.[ch]
  - debug.[ch]
  - deployment_unit.[ch]
  - dimark_globals.h
  - dimclient.[ch]
  - du_transfer.[ch]
  - eventcode.[ch]
  - filetransfer.[ch]
  - ftcallback.[ch]
  - list.[ch]
  - methods.[ch]
  - paramaccess.[ch]
  - paramconvenient.[ch]
  - parameter.[ch]
  - serverdata.[ch]
  - si_transfer.[ch]
  - utils.[ch]
  - vouchers.[ch]
- STUN_client – stores files that implement STUN client.
  - STUN_client.[ch]
  - STUN_packet.[ch]
- tmp – stores temporary files. Doesn't exist on the "fresh" client.
- TR-098 – stores initial client parameter configuration file for Internet Gateway Device Data Model.
  - data-model.xml – sample of client parameter configuration file in .xml format.
- TR-106 – stores initial client parameter configuration file for TR-069-Enabled Devices.

- data-model.xml – sample of client parameter configuration file in .xml format.
- apply_gSoap_patch.sh
- ca.crt – cacert file to store trusted certificates (needed to verify server).
- ChangeLog.txt – describes all the changes made to Client in each release version.
- clientEditor.jar – allows the user to edit *.param files.
- config.txt – stores file that will be uploaded if Upload argument FileType is "1 Vendor Configuration File".
- configure – file with the help of which a `Makefile` is created. Using this file the operator configures part of the Client compilation flags.
- configure.in
- conv.jar – converts old format of client parameter configuration files to new one (dps.param to data-model.xml) and vice versa.
- conv-util.c – allows the client to read *.xml files and treat them as parameter configuration files.
- data-model.xml – sample of client parameter configuration file in .xml format.
- dimclient.conf – external configuration file for setting [InternetGateway]Device.ManagementServer.URL, [InternetGateway]Device.ManagementServer.Username, and [InternetGateway]Device.ManagementServer.Password. Log settings are also specified here.
- download.dwn – this file stores the downloaded file and is created if there is no other way to find out the name of downloading file. First the system checks whether the value of the TargetFileName argument of the Download RPC used to download this file was specified. If not, the `getDefaultDownloadFilename()` function is called. This function checks whether the download file name can be extracted from the download URL. If not, the function returns the file name that is stored in DEFAULT_DOWNLOAD_FILE in dimclient.c. Currently this name is "download.dwn". Download.dwn file doesn't exist on "fresh" client.
- host-if.c
- log.config – allows to configure Dimark Client log. User can assign the following logging levels DEBUG, INFO, WARN and ERROR.
- logfile.txt – stores file that will be uploaded if Upload argument FileType is "2 Vendor Log File".
- Makefile.in
- start.sh – starts Dimark Client. See the start.sh script to find out how the Client needs to be started and how its target data directory should be prepared.

Runtime Database

The client provides the option to modify the way it stores data.

The default implementation that the client is shipped with, uses a new runtime DB – SQLite.
The default settings imply that during client configuration the operator chooses default client configuration or uses custom configuration, but with `without-sqlitedb` flag disabled.
In this case, the data will be stored in tmp/parameters.db.

The second implementation that can be set, uses the old format of runtime DB.
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 old 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. The file works like marker and is deleted after FactoryReset and does not exist before the client first start. Otherwise it persists.
EVENT_STORAGE_FILE	./tmp/eventCode.dat	Event Storage of the client that must survive a reboot.
DEFAULT_DOWNLOAD_FILE	./download.dwn	Default download file. File stores the downloaded file and is created if there is no other way to find out the name of downloading file (neither from the target name specified on Auto configuration server, nor from the download URL).
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).
PERSISTENT_TRANSFERLIST_DIR	./tmp/filetrans/	Directory where unfinished transfers are stored. When the Client is rebooted data about transfers is read from this directory.

Define	Typical Value	Usage
PERSISTENT_DU_ ENTRYLIST_DIR	./tmp/duentries/	Directory where information about deployment units is stored. When the Client is rebooted data about deployment units is read from this directory.
PERSISTENT_SI_ ENTRYLIST_DIR	./tmp/sientries/	Directory where data about unperformed schedule informs is stored. When the Client is rebooted data about schedule informs is read from this directory.

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

6. Porting the Client

Before porting the Dimark Client onto your target platform, it is strongly recommended to build the client on a regular Linux machine (i.e. Red Hat Enterprise Linux 5+, SUSE Linux Enterprise 10+, etc) to familiarize yourself with the output of the client and to establish a point of reference before starting the porting effort.

> Starting from Client v. 4.0 in order to run and compile Dimark Client user should check the presence and version of the `crypto`, `openssl`, `libxml2`, `sqlite3` and `uuid` 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 `openssl` and `crypto` shared libraries.

2. Run `rpm -qa | grep libxml2` command.

   It checks the presence of `xml2` shared library.

3. Run `rpm -qa | grep sqlite3` command.

   It checks the presence of `sqlite3` shared library.

4. Run `rpm -qa | grep uuid` command.

   It checks the presence of `uuid` shared library. The `uuid` library is used to generate unique identifiers for deployment unit operations. Each deployment unit operation contains an argument called UUID, which enables an ACS to uniquely identify a deployment unit across CPE. Normally the UUID is generated by the ACS, but there are circumstances when a CPE must generate its own UUID and that is why the `uuid` library is used by the Client.

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-vdso.so.1 =>  (0x00007fff7efff000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fa004cb6000)
libcrypto.so.1.0.0 => not found <--------> Library is not available
libssl.so.1.0.0 => not found<--------> Library is not available
libsqlite3.so.0 => /usr/local/lib/libsqlite3.so.0
```

```
(0x00007fa00440a000)
libuuid.so.1 => /lib64/libuuid.so.1 (0x00007fa004205000)
libc.so.6 => /lib64/libc.so.6 (0x00007fa003e76000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa004ed3000)
libdl.so.2 => /lib64/libdl.so.2 (0x00007fa003c72000)
libz.so.1 => /lib64/libz.so.1 (0x00007fa003a5a000)
```

Compiler Flags for the Client

Please note: for compilation development versions of the libraries are required:

```
libxml2-devel, openssl-devel, sqlite3-devel, uuid-devel.
```

In case if the libraries are located in another folder, perform the following command to create a symbolic link:

```
ln -s /usr/include/libxml2/libxml /usr/include/libxml
```

Please consider that Client memory consumption depends on number of flags you have set. By disabling methods that you are not going to use you can reduce memory consumption up to 50-70%.

The data stated below was received on the test platform openSUSE 11.4 x64, core 2.6.37.6-0.11, glibc version 2.11.3, gcc version 4.5.1.

VmHWM: 7448 kB
VmRSS: 7420 kB
Threads: 14

This data is approximate because the sizing results depend on the operating system, compiler version, dimclient loading.

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

Please, check that configuration and compiler flags are set as needed!

Configuration Flags

Configuration Flag	Location	Usage
with-clientsslauth	Configure	Enables SSL Authentication through certificates. Server checks client certificate. For correct operation: 1) Generate a couple of SSL Client certificates. 2) Either add public certificate to ACS trusted certificates or get the Client certificate signed by the trusted certificate authority (ACS should be aware of this certificate authority).
with-ipv6	Configure	Configures client with IPV6 support. By default the client operates only with IPV4, but when compiled with this flag it has both IPV4 and IPV6 support.
with-serversslauth	Configure	Enables SSL Authentication through certificates. Client checks server certificate. For correct operation: 1) Generate a couple of SSL ACS certificates. 2) Either add public certificate to Client trusted certificates or get the ACS certificate signed by the trusted certificate authority (Client should be aware of this certificate authority).
with-stunclient	Configure	Enables STUN client and activates UDP Connection Request listener.
with-device_root	Configure	Results in the client using an object model starting with "Device."
with-tr181	Configure	Results in the client using TR-181 data model starting with "Device.
without-openssl	Configure	OpenSSL support is present by default. This flag configures Client without SSL support.
without-sqlitedb	Configure	Defines that client will work with old format of runtime DB instead of using new SQLite DB.

Compiler Flags in dimark_globals.h file

Compiler Flag	Location	Usage
FILETRANSFER_STORE_STATUS	dimark_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	dimark_globals.h	Defines if the client should update the NumberOfEntries of an Object during an AddObject or DeleteObject.
HAVE_ATMF5_DIAGNOSTICS	dimark_globals.h	Enables / disables ATMF5 diagnostics.

Compiler Flag	Location	Usage
HAVE_AUTONOMOUS_DU_STATE_CHANGE_COMPLETE	dimark_globals.h	Enables / disables AutonomousDUStateChangeComplete method.
HAVE_AUTONOMOUS_TRANSFER_COMPLETE	dimark_globals.h	Enables / disables AutonomousTransferComplete method.
HAVE_CANCEL_TRANSFER	dimark_globals.h	Enables / disables CancelTransfer method.
HAVE_CONNECTION_REQUEST	dimark_globals.h	Enables / disables ConnectionRequest support.
HAVE_DEPLOYMENT_UNIT	dimark_globals.h	Enables / disables deployment unit support.
HAVE_DIAGNOSTICS	dimark_globals.h	Enables / disables all diagnostics.
HAVE_DOWNLOAD_DIAGNOSTICS	dimark_globals.h	Enables / disables DownloadDiagnostics method. Does not concern Download method.
HAVE_FACTORY_RESET	dimark_globals.h	Enables / disables FactoryReset method.
HAVE_FILE	dimark_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	dimark_globals.h	Enables / disables both Download and DownloadDiagnostics methods.
HAVE_FILE_UPLOAD	dimark_globals.h	Enables / disables both Upload and UploadDiagnostics methods.
HAVE_FILE_SCHEDULE_DOWNLOAD	dimark_globals.h	Enables / disables both ScheduleDownload method.
HAVE_GET_ALL_QUEUED_TRANSFERS	dimark_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	dimark_globals.h	Enables / disables GetAllQueuedTransfers method.
HAVE_HOST	dimark_globals.h	Enables / disables host interface support.
HAVE_IP_PING_DIAGNOSTICS	dimark_globals.h	Enables / disables IP Ping diagnostics.
HAVE_KICKED	dimark_globals.h	Enables / disables Kicked method.

Compiler Flag	Location	Usage
HAVE_OPTIONAL	dimark_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	dimark_globals.h	Enables / disables RequestDownload method.
HAVE_RPC_METHODS	dimark_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	dimark_globals.h	Enables / disables ScheduleInform method.
HAVE_TRACE_ROUTE_DIAGNOSTICS	dimark_globals.h	Enables / disables TraceRoute diagnostics.
HAVE_UDP_ECHO	dimark_globals.h	Enables / disables UDP Echo diagnostics (TR-143 standard).
HAVE_UPLOAD_DIAGNOSTICS	dimark_globals.h	Enables / disables UploadDiagnostics method. Does not concern Upload method.
HAVE_VOUCHERS_OPTIONS	dimark_globals.h	Enables / disables both SetVouchers and GetOptions methods. 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	dimark_globals.h	Enables / disables WAN DSL diagnostics.
NO_LOCAL_REBOOT_ON_CHANGE	dimark_globals.h	Disables automatic reboot when changes are made to parameters that normally require a reboot. 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). 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.

Compiler Flag	Location	Usage
WITH_PRINT_MEM_MSG	dimark_ globals.h	Enables / disables printing messages about manual allocating and freeing memory to screen-log.
WITH_SOAPDEFS_H	dimark_ globals.h	Enables / disables soapdefs.h file. This flag should always be present.

Compiler Flags in Makefile.in file

Compiler Flag	Location	Usage
ACS_REGMAN	Makefile.in	Enables 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.
SOAP_DISPLAY_MSG	Makefile.in	All messages (post and get requests) are displayed on the screen instead of saving into the debug.log and test.log (files are created when the Client is started).
WITH_COOKIES	Makefile.in	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_SSL_RESUMPTION	Makefile.in	After the first SSL handshake the subsequent sessions begin with a short SSL handshake until the Client is rebooted. After the reboot the Client again performs the standard handshake and proceeds with short handshakes again.
WITHOUT_DEBUG=TRUE	Makefile.in	Defines that compilation of client debug information will be disabled and it will not be included to code.

Building the Client on Red Hat Enterprise Linux

This section describes how to build the client code on a Linux system. While this section uses Red Hat Enterprise Linux distribution in the examples, it will work on all Linux distributions as well.

Required Additional Components

In order to build the client you have to obtain gSOAP (gSOAP Version 2.8.8 is required).

Please, download gSoap from http://sourceforge.net/projects/gsoap2/files/gSOAP/ (gSOAP Version 2.8.8). Remember, that the commercial license agreement for gSOAP Version 2.8.8 (standard commercial edition) should be obtained from Genivia (http://www.genivia.com/Products/gsoap/contract.html).

Please consult ChangeLog*.txt and README Client files to check if any changes has been made concerning required gSOAP version.

It is also assumed that OpenSSL is already installed on your platform. If not, obtain any version of OpenSSL from 0.9.7f through 0.9.8i and follow the directions for compiling and installing.

Extract the Client

Create the user/tr069 directory and extract the client cod into it. Execute the following command to extract the client code:

```
tar xf <path to>/DPSClient-<Version>.tgz
```

This will extract the client source code into user/tr069 folder. You will as well need to copy some gSOAP files to the `gsoap` and to the `plugin` directories. To support TR-069, those files should be modified with the help of `apply_gSoap_patch.sh` script.

Compilation Process

Compilation process can be divided into the following steps:

- Client configuration;
- Client compilation;
- Starting the Client.

Client Configuration

Before starting the compilation process, the Dimark Client should be properly configured.
Configuration process consists of two major actions:

- configuring Dimark Client flags;
- patching Dimark Client with gSoap files.

Each action can be performed independently, but both should be finished before compiling the Client.

Configuring Dimark Client flags

Dimark Client flags are no longer edited in `Makefile`.

`Makefile` even doesn't exist on the "fresh" client.

Starting from version 4.1 Dimark Client flags are configured with the help of the following sequence:

1. Download the Dimark Client.
2. Go to the project root folder and run `configure` file:

./configure

This command creates a Makefile with default flags settings, where:

without-openssl – flag is disabled. Client uses OpenSSL;

without-sqlitedb – flag is disabled. Client uses new runtime DB – SQLite.

This variant of Client configuration is strongly recommended unless user needs to customise Client flags.

If operator needs to customise flag settings, the help data will assist in getting more information about possible variants of `Makefile` creation:

./configure --help

To adjust configure file, the following pattern should be used:

./configure --flag1 --flag2 ...--flagN

Please note: flag names should be typed using only small letters!

`configure` file (either with default configuration or with changes) uses `Makefile.in` file to automatically create `Makefile` file in the root directory. Along with `Makefile`, two additional files are created `config.status` and `config.log`.

3. When the `Makefile` is created the project is ready to be compiled with the help of `make` command.
4. If any changes are needed to be done in compilation/building settings, the operator should again run

./configure

or

./configure --flag1 --flag2 ...--flagN

Then `make clean` and `make`.

Patching Dimark Client with gSoap files

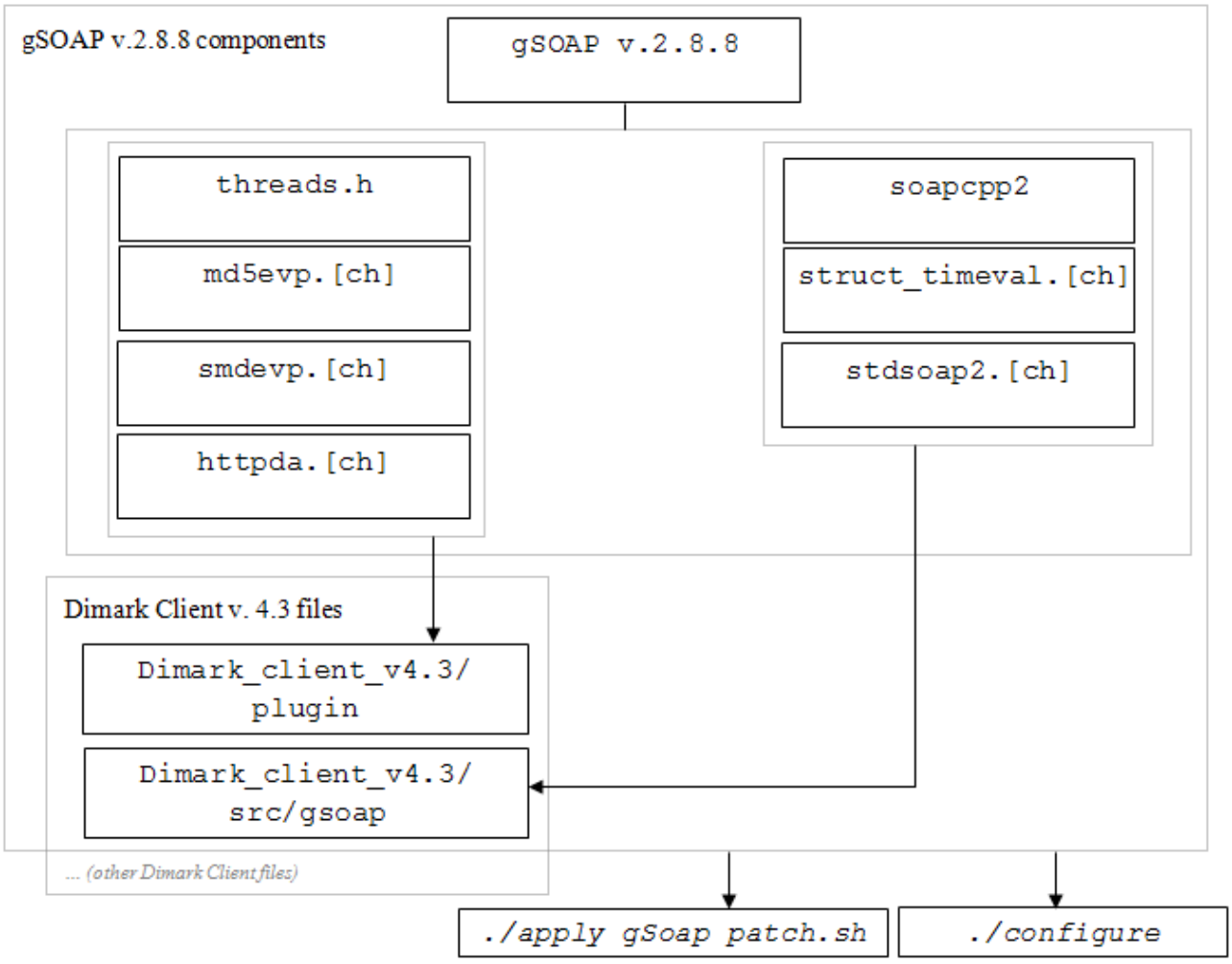
1. Download gSoap v.2.8.8. from official site.
2. Copy the following gSoap files to ./plugin directory:
   - threads.h
   - md5evp.[ch]
   - smdevp.[ch]
   - httpda.[ch]
3. Copy the following gSoap files to ./gsoap directory:
   - soapcpp2
   - struct_timeval.[ch]
   - stdsoap2.[ch]

4. Run

`./apply_gSoap_patch.sh`

5. When the Client is patched with gSoap files, the project is ready to be compiled with the help of make command.

The schema of both Client configuration steps can be seen below:



Client Compilation

To perform the client compilation/building go to the Dimark client source directory. Be sure that the configuration step described above was performed properly (the client was patched with gSoap files and the needed flags were configured). When ready to compile, type: `make`

The Dimark client will first call the soapcpp2 process to compile the gSoap directives to produce the necessary files to continue compilation. Some warning messages will be produced by soapcpp2. These are normal and can be ignored. The compilation will continue. When complete the following binary files will be present: `host-if`, `dimclient` and `conv_util`.

You may need other compilation commands:

`make all` or `make` – project compilation;

`make release` – release preparation. Performs the `make clean` command and additionally deletes all files of the revision control system;

`make clean` – deletion of the intermediate files created during compilation and execution;

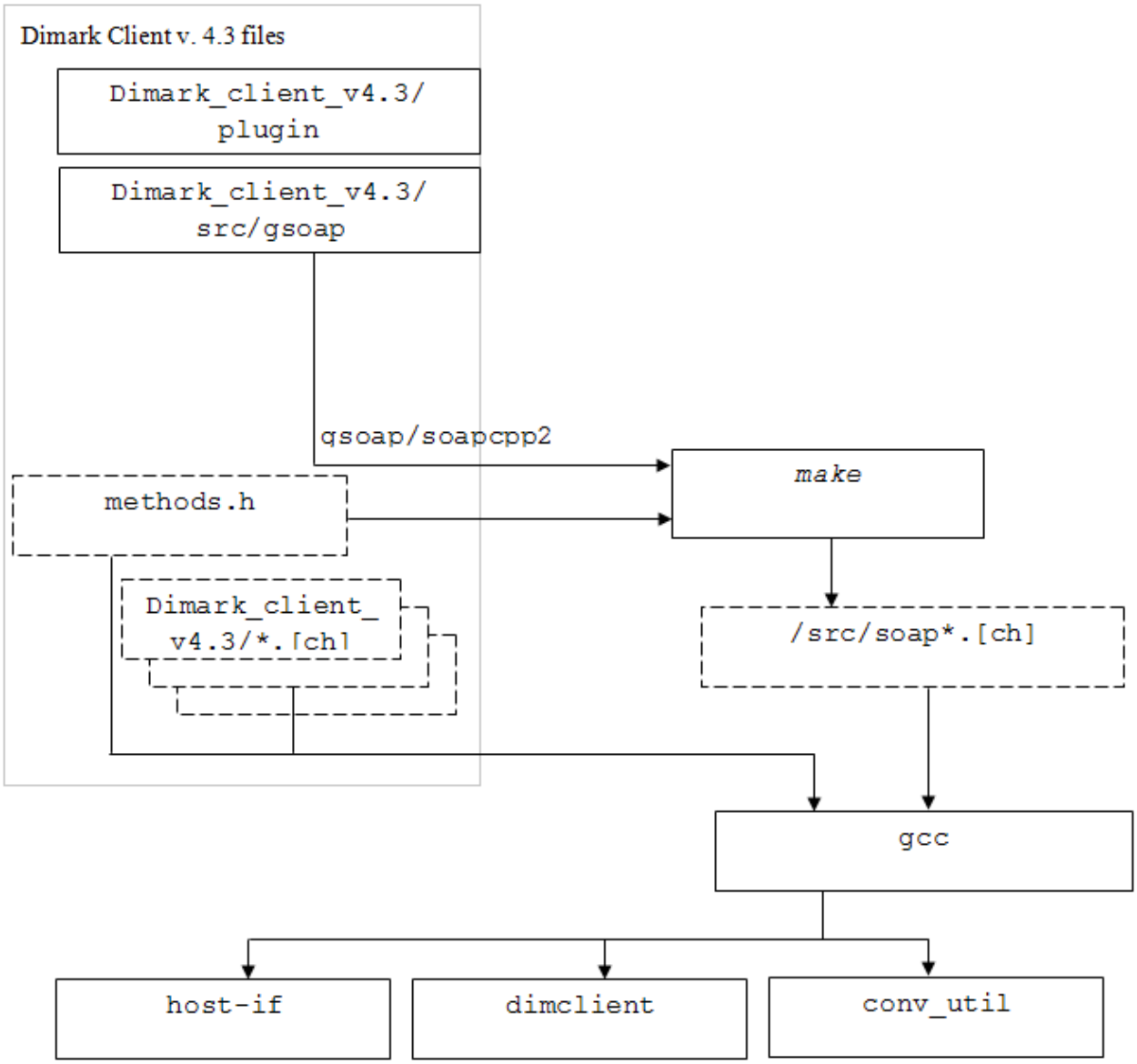
`make install` – project installation;

`make dimclient` – dimclient application compilation;

`make conv-util` – conv-util application compilation;

`make host-if` – host-if application compilation.

The schema of Client compilation can be seen below:

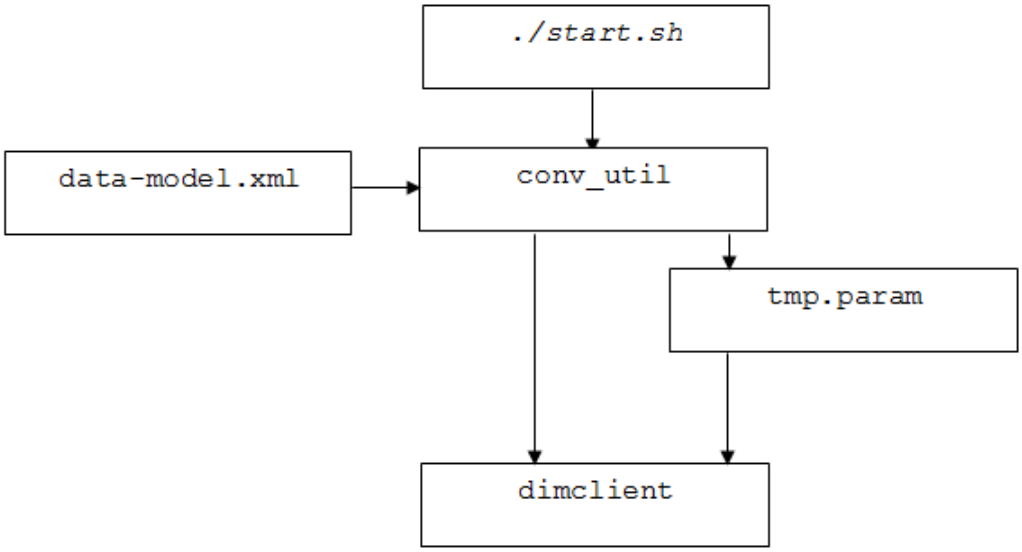


Starting the Client

See the start.sh script for how the client needs to be started and its target data directory prepared.

Run `start.sh` and the `dimclient` will be run automatically.

The schema of starting the Client can be seen below:



Please note: Although *.param is considered deprecated, it's still used internally because of much smaller file size (often convenient in production).

Dimark client bootstrap procedure:
data-model.xml -> conv-util -> tmp.param -> dimclient -> runtime parameters DB.

7. 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 configuration 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.

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:

Parameter Structure	Example of Parameter Name	data-model.xml representation

Top-level object. Parameter	`Device.` `DeviceSummary`	`<object base="Device."...>`
		`  <parameter base="DeviceSummary"...>`
		`  <syntax>`
		`  <string/>`
		`  <default type.../>`
		`  </syntax>`
		`  </parameter>`
		`</object>`

Top-level object. Second-level object. Parameter	`Device.Info.` `ManufacturerOUI` and `Device.Info.` `SerialNumber` (this example also shows how to represent parameters which belong to same object)	`<object base="Device.Info." ...>`
		`  <parameter base="ManufacturerOUI" ...>`
		`  <syntax>`
		`  <string/>`
		`  <default type.../>`
		`  </syntax>`
		`  </parameter>`
		`.....`
		`  <parameter base="SerialNumber" ...>`
		`  <syntax>`
		`  <string/>`
		`  <default type.../>`
		`  </syntax>`
		`  </parameter>`
		`</object>`

Parameter Structure	Example of Parameter Name	data-model.xml representation
Top-level object. Second-level object. Third-level object. Parameter	Device.LAN.Stats. ConnectionUpTime	<object base="Device.LAN.Stats." ...> <parameter base="ConnectionUpTime" ...> <syntax> <unsignedInt/> <default type.../> </syntax> </parameter> </object>

The full examples are as follows:

Top-level object.Parameter

```
<object base="Device." access="readOnly" minEntries="1" maxEntries="unbounded">
   <parameter base="DeviceSummary" access="readOnly" notification="3"
   alwaysInclude="true" maxNotification="2" instance="0" reboot="1" initIdx="-1"
   getIdx="0" setIdx="-1">
       <syntax>
           <string/>
               <default type="factory" value="Device:1.0[](Baseline:1,
               IPPing:1),STBService:1.0[](Baseline:1)"/>
       </syntax>
   </parameter>
</object>
```

Top-level object.Second-level object.Parameter

```
<object base="Device.Info." access="readOnly" minEntries="1" maxEntries="unbounded">
   <parameter base="ManufacturerOUI" access="readOnly" notification="0"
   maxNotification="2" instance="0" reboot="1" initIdx="-1" getIdx="0" setIdx="-1">
       <syntax>
           <string/>
               <default type="factory" value="999999"/>
       </syntax>
   </parameter>
.......
```

```
    <parameter base="SerialNumber" access="readOnly" notification="0"
    maxNotification="2" instance="0" reboot="1" initIdx="-1" getIdx="0" setIdx="-1">
        <syntax>
            <string/>
                <default type="factory" value="1234567890-Device"/>
        </syntax>
    </parameter>
</object>
```

Top-level object.Second-level object.Third-level object. Parameter

```
<object base="Device.LAN.Stats." access="readOnly" minEntries="1"
maxEntries="unbounded">
    <parameter base="ConnectionUpTime" access="readOnly" notification="0"
    maxNotification="1" instance="0" reboot="0" initIdx="1" getIdx="1" setIdx="-1">
        <syntax>
            <unsignedInt/>
                <default type="factory" value="0"/>
        </syntax>
    </parameter>
</object>
```

where:

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

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

`SerialNumber` – parameter name of Device.ManagementServer TR-069 parameter;

`notification="0" maxNotification="2" reboot="1" initIdx="1" getIdx="1" setIdx="1"` – details that characterize 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.

Data Type

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

```
<object ....>
    <parameter ... >
      <syntax>
```

```
        <string/>
         <default type.../>
      </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 11 in the dps.param file);
- Integer (formerly 12 in the dps.param file);
- UnsignedInteger (formerly 14 in the dps.param file);
- Boolean (formerly 16 in the dps.param file);
- DateTime (formerly 10 in the dps.param file);
- Base64 (formerly 15 in the dps.param file);
- UnsignedLong 20;
- HexBinary 21;
- Int.

Types for DefaultObject (formerly 110, 111, 113, 115, 109, 114, 119, 120 in the dps.param file) are represented as corresponding data types for `default type="object"`.

Access List
Access List - list of access rights (i.e. Subscriber).

Tag access list should be located before <syntax> tag as in the following example. The example of access rights has two entries - x and y:

```
<parameter ...>
    <accessList>
       <entry>x</entry>
       <entry>y</entry>
    </accessList>
    <syntax>
       <string/>
       <default ...>
    </syntax>
</parameter>
```

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

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

Default Value

Default Value - the initial value of the parameter. Default Value is stated in the <syntax> tag below the data type tag and is presented as follows:

```
<parameter base="URL" access="readOnly" notification="0"
maxNotification="2" instance="0" reboot="1" initIdx="1" getIdx="1"
setIdx="1">
 <syntax>
  <string/>
   <default type="factory" value="http://test.dimark.com:8080/dps/
TR069"/>
 </syntax>
</parameter>
```

Details that characterise the TR-069 Parameter.

An example from a TR-111 Device is:

```
<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>
```

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. 0 = no notification; 1 = passive notification (if parameter with this notification type is changed it will be included into the next regular Inform). Parameter change is checked according to the period stated in DefaultActiveNotificationThrottle parameter. Parameters are reported in the next regular Inform. Please mention, if DefaultActiveNotificationThrottle is not set or its value is 0, neither passive notification nor active will work. In this case only DefaultActiveNotificationThrottle will be checked each 10 seconds; 2 = active notification (an Inform will be initiated when parameter with this notification type is changed). Parameter change is checked according to the period stated in DefaultActiveNotificationThrottle. Please mention, if DefaultActiveNotificationThrottle is not set or its value is 0, neither passive notification nor active will work. In this case only DefaultActiveNotificationThrottle will be checked each 10 seconds; 3 = forced notification (parameter with this notification type will be always included into 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). 0 = Reboot 1 = No reboot required
initIdx	Allows calling an initialization function when the data is loaded from configuration file data-model.xml. Set initIdx to -1 if no initialization is required. In case the initIdx of the parameter was set to -1, the parameter will not be stored in runtime database. Thus, Get and Set methods for this parameter cannot be executed.
getIdx	The index number that was given to this parameter in the getArray[] of paramaccess.c. Set to -1 if the parameter does not have a Get method, and is Write Only (i.e. Passwords). An entry of 0 reads the value from RAM. An entry of 1 reads the data from the persistent layer. getIdx can possess any number declared in the setArray[].
setIdx	The index number that was given to this parameter in the setArray[] of paramaccess.c. An entry of -1 treats the parameter as Read Only. An entry of 0 causes the value to be read from the CPE. An entry of 1 writes the data to the persistent layer. setIdx can possess any number declared in the setArray[].

Operating with `getIdx` and `setIdx` user can change write and read permissions of the parameter. Possible combinations are as follows:

`getIdx="-1"`
`setIdx` any value, except `"-1"`
Parameter is WriteOnly.

`getIdx` any value, except `"-1"`
`setIdx="-1"`
Parameter is ReadOnly.

`getIdx` any value, except `"-1"`
`setIdx` any value, except `"-1"`
Parameter is ReadWrite.

`getIdx="-1"`
`setIdx="-1"`
Parameter is neither writable, not readable.
There are no parameters in the TR standards which can not be read and written simultaneously. This combination is pointless and shouldn't be used, because it results in error.

Creating Multiple Objects

TR-069 allows definition of Objects that appear multiple times under a single parent object. This release of the client supports a "hidden" instance of the object that can be used to define default values and to initiate newly created object.

Definition of such object is listed below (TR-104 Voice Profile). When a new Object is created from the ACS the Object Instance zero (0) is taken as initial value of new object instances. If there would be no Instance 1 defined, the ACS would not be able to see the details of the TR-104 Voice Profile elements as no visible instance would be available.

The following example is an excerpt from TR-104 defining a Profile and Line association. This profile allows dynamic creation of both Profiles and Lines.

```
<object base="Device.Services.VoiceService.Profile.{i}."
access="readOnly" minEntries="1" maxEntries="unbounded">
<parameter base="Enable" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="1" initIdx="3" getIdx="1"
setIdx="1">
<syntax>
<boolean/>
```

```
<default type="object" value="0"/>
</syntax>
</parameter>
<parameter base="Name" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="2" getIdx="1"
setIdx="-1">
<syntax>
<string/>
</syntax>
</parameter>
</object>

<object base="Device.Services.VoiceService.Profile.{i}.Line.{i}."
access="readOnly" minEntries="1" maxEntries="unbounded">
<parameter base="Enable" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="1" getIdx="1"
setIdx="">
<accessList>
<entry>0</entry>
</accessList>
<syntax>
<boolean/>
</syntax>
</parameter>
<parameter base="DirectoryNumber" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="1" getIdx="1">
<syntax>
<string/>
</syntax>
</parameter>
</object>
```

This sample only defines a subset of the required TR-104 parameters to illustrate how nested objects can be created. This definition allows the client to create a new profile, that won't have any visible line information underneath. The line would have to be created with another AddObject call from the ACS.
In case the CPE defines already an existing element, it as well must define the Instance number zero (0) for those elements in the object tree that can be dynamically instantiated. The following code is an example of such definition in TR-104:

```
<object base="Device.Services.VoiceService.Profile.{i}."
access="readOnly" minEntries="1" maxEntries="unbounded">
<parameter base="Enable" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="1" initIdx="3" getIdx="1"
setIdx="1">
<syntax>
<boolean/>
<default type="object" value="0"/>
</syntax>
</parameter>
<parameter base="Name" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="2" getIdx="1"
setIdx="-1">
<syntax>
<string/>
</syntax>
</parameter>
</object>

<object base="Device.Services.VoiceService.Profile.{i}.Line.{i}."
access="readOnly" minEntries="1" maxEntries="unbounded"/>

<object base="Device.Services.VoiceService.Profile.1."
access="readOnly" minEntries="1" maxEntries="unbounded">
<parameter base="Enable" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="0" getIdx="0"
setIdx="">
<accessList>
<entry>0</entry>
</accessList>
<syntax>
<boolean/>
</syntax>
</parameter>
<parameter base="Name" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="2" getIdx="1"
setIdx="-1">
```

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

<object base=”Device.Services.VoiceService.Profile.{i}.Line.{i}.”
access=”readOnly” minEntries=”1” maxEntries=”unbounded”>
<parameter base=”Enable” access=”readOnly” notification=”0”
maxNotification=”0” instance=”0” reboot=”0” initIdx=”1” getIdx=”1”
setIdx=””>
<accessList>
<entry>0</entry>
</accessList>
<syntax>
<boolean/>
</syntax>
</parameter>
<parameter base=”DirectoryNumber” access=”readOnly” notification=”0”
maxNotification=”0” instance=”0” reboot=”0” initIdx=”1” getIdx=”1”>
<syntax>
<string/>
</syntax>
</parameter>
<parameter base=”Enable” access=”readOnly” notification=”0”
maxNotification=”0” instance=”0” reboot=”0” initIdx=”1” getIdx=”1”
setIdx=””>
<accessList>
<entry>0</entry>
</accessList>
<syntax>
<boolean/>
</syntax>
</parameter>
<parameter base=”DirectoryNumber” access=”readOnly” notification=”0”
maxNotification=”0” instance=”0” reboot=”0” initIdx=”1” getIdx=”1”>
<syntax>
```

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

<object base="Device.Services.VoiceService.Profile.1.Line.1."
access="readOnly" minEntries="1" maxEntries="unbounded">
<parameter base="Enable" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="0" getIdx="0"
setIdx="">
<accessList>
<entry>0</entry>
</accessList>
<syntax>
<boolean/>
</syntax>
</parameter>
<parameter base="DirectoryNumber" access="readOnly" notification="0"
maxNotification="0" instance="0" reboot="0" initIdx="0" getIdx="0">
<syntax>
<string/>
</syntax>
</parameter>
</object>
```

The difference between the first and second example is that the second example already defines a profile instance 1 with an associated Line (instance 1). In each case the ACS can create additional profiles and lines underneath those objects dynamically.

The following table represents the definitions of details that characterise objects:

Detail Element	Description
maxEntries	Attribute is used by multi-instance objects (type 101). Attribute reflects maximum number of instances that can be created for the object. "unbounded" value means that the maximum number of instances is not limited.

Please note: If deprecated datamodel format dps.param is used, user should add two more attributes for multi-instance objects (type 101): `minEntries` and `maxEntries` in the end of the line. In the following sample the last attributes are `minEntries="1" maxEntries="unbounded"`:

```
InternetGatewayDevice.Layer3Forwarding.Forwarding.;101;0;0;0;0;0;
0;;;1;unbounded
```

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/dimark_globals.h that defaults to port 8081 – Hosthandler interface port) 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
```

Please 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>
```

8. Additional Parameters

Dimark Client specific 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).

Vendor-Specific Parameters

A vendor may have additional vendor-specific parameters and objects.

The name of a vendor-specific parameter or object not contained within another vendor-specific object must have the form:

```
X_<VENDOR>_VendorSpecificName
```

In this definition <VENDOR> is a unique vendor identifier, which may be either an OUI or a domain name. The OUI or domain name used for a given vendor-specific parameter must be one that is assigned to the organization that defined this parameter (which is not necessarily the same as the vendor of the CPE or ACS).

An OUI is an organizationally unique identifier as defined in [15]. A domain name must be upper case with each dot (".") replaced with a hyphen or underscore.

The VendorSpecificName must be a valid string and must not contain a "." (period) or a space character.
The full path name of a vendor-specific parameter or object must not exceed 256 characters in length.

Below are some example vendor-specific parameter and object names:

```
InternetGatewayDevice.UserInterface.X_012345_AdBanner
InternetGatewayDevice.LANDevice.1.X_012345_LANInfraredInterfaceConfi
g.2.Status
```

To add a Vendor-Specific Parameter into data-model.xml file user should edit paramaccess.c and parameter.c files.

1. In paramaccess.c file set name and unique ID for Get of Set method and add them to `getArray[]` or `setArray[]` as shown in the following example:

```
Func getArray[] = {
{124, &getUptime},
};
```

or

```
Func setArray[] = {
{2, &setInt},
};
```

where:
`124` and `2` – unique IDs of Get and Set functions. Mention,that some of the index entries are already assigned by the client and cannot be changed (-1; 0; 1);
`getUptime` and `setInt` – names of Get and Set functions.

2. In paramaccess.c file add function body for new Get and Set functions.
For example:

```
static int
getUptime (const char *name, ParameterType type,
ParameterValue *value)
{
struct sysinfo info;
sysinfo (&info);
value->out_uint = (unsigned int) info.uptime;
return OK;
}
```

or

```
static int
setInt (const char *name, ParameterType type, ParameterValue
*value)
```

```
{
DEBUG_OUTPUT (
dbg (DBG_ACCESS, "SetInt %s %d\n", name, in);
)
return OK;
}
```

3. In parameter.c file find the `loadParameters (bool bootstrap)` function. Function contains the default mapping of Get and Set functions.
4. In the `loadParameters (bool bootstrap)` function find the following part:

```
// Load the initial Parameters with the whole Structure
ret = loadInitialParameterFile( (newParam *)&newParam1 );
paramBootstrap = false;
dimClientId.ival = 83620;
strcpy( buf, DEVICE_INFO )
strcat( buf, "X_DIMARK_COM_ClientID" );
newParameter( buf, IntegerType, 0, NoReboot,
NotificationAllways, NotificationAllways, -1, 0, -1, "",
&dimClientId, true);
dimClientVersion.cval = "Agent3.0";
strcpy( buf, DEVICE_INFO );
strcat( buf, "X_DIMARK_COM_Version" );
newParameter( buf, StringType, 0, NoReboot,
NotificationAllways, NotificationAllways, -1, 0, -1, "",
&dimClientVersion, true );
```

5. Add code according to the following example and specify `<getIdx>` and `<setIdx>` if you want to add a vendor-specific parameter and its value:

```
strcpy( buf, Vendor-Specific_Parameter_PATH );
strcat( buf, "Vendor-Specific_Parameter_NAME1" );
newParameter( buf, IntegerType, 0, NoReboot,
NotificationAllways, NotificationAllways, <initIdx >, <getIdx>,
<setIdx>, "", &dimClientId, true);
dimClientVersion.cval = "Vendor-Specific_Parameter_VALUE";
```

Or if you don't intend to set value for a vendor-specific parameter, add code according to the following

example and specify <getIdx> and <setIdx>:

```
strcpy( buf, Vendor-Specific_Parameter_PATH );
strcat( buf, "Vendor-Specific_Parameter_NAME1" );
newParameter( buf, IntegerType, 0, NoReboot,
NotificationAllways, NotificationAllways, <initIdx >, <getIdx>,
<setIdx>, "", &dimClientId, true);
dimClientVersion.cval = "Vendor-Specific_Parameter_VALUE";
```

Please mention that IntegerType, 0, NoReboot, NotificationAllways, NotificationAllways, <initIdx >, <getIdx>, <setIdx>, "" contain details that characterize the TR-069 Parameter – data type, numbet of inctances for Objects that have multiple instances, reboot, notification, maxNotification, initIdx, getIdx, setIdx, access list. For further details please see the "XML Configuration File Description" section of this document.

6. Perform the compilation.

9. 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.

10. Implementing Storage Environment

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)
```

Before using the `getDefaultDownloadFilename()` function, the system checks whether the value of the TargetFileName argument of the Download RPC used to download this file was specified. If not, the `getDefaultDownloadFilename()` function is called. This function checks whether the download file name can be extracted from the download URL. If not, the function returns the file name that is stored in DEFAULT_DOWNLOAD_FILE in dimclient.c. Currently this name is “download.dwn”.

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)
```

11. 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).

```
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.

12. Diagnostics Support

Dimark Client performs tracking and processing of diagnostics in the separate thread. Diagnostics thread is performed simultaneously with the main thread. If the diagnostics were assigned they are processes according to FIFO principles.

Dimark Client supports following TR-143 diagnostics:
Download Diagnostics utilizing FTP transport and Download Diagnostics utilizing HTTP transport [16, 17].
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 [16, 17].
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 set by appropriate parameter (default 8088 – port for UDP echo diagnostics listening). When a datagram is received, the data from it is sent back in an answering datagram. UDP Echo Plus support is implemented.

For further TR-143 diagnostics details please see TR-143, Enabling Network Throughput Performance Tests and Statistical Monitoring, Broadband Forum Technical Report [10].

Dimark Client supports following TR-098 diagnostics:
IP Ping Diagnostics is being used for calculating the number of successful and failed pings and measuring average, minimum and maximum response time while pinging the specified host during the ping test.

Trace Route Diagnostics for tracing the routing path to a given destination machine. This is a very important diagnostic tool for administrators and service providers. Its main purpose is to find out the route IP packets take to reach a specific destination.
Before building the client integrator can choose a protocol which will be used by the Trace Route Diagnostics (UDP, ICMP, TCP). The changes should be made in the following line of `diagParameter.c` file:

```
TraceRouteProtocol traceRouteProtocol = UDP;  // TODO: It is needed
to choose a traceroute protocol { ICMP, UDP, TCP }.
```

Additionally the integrator can specify the path to traceroute command (see `diagParameter.c` file). Currently the following path is used:

```
#define TRACE_ROUTE_COMMAND "/usr/sbin/traceroute"  //traceroute
command (or path + command), specified by OS
```

For further TR-098 diagnostics details please see TR-098, Internet Gateway Device Data Model for TR-069 (Amendment 2), Broadband Forum Technical Report [2].

13. 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 on which CPE is listening to Kicked requests.
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.

14. 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://IP:port/command=Download&type=1 Firmware Upgrade
Image&url=http://URL&username=user&password=name&delay=10000&size=2
4024&target=target&success=success&failure=failure
```

Sample of Upload method invocation:

```
http://IP:port/command=Upload&type=2 Vendor Log File&url=http://URL
&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).

Please 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://IP:port/command=Download&url=http://URL
```

15. Starting Multiple Client Instances

Several Clients can be started on one machine (using one IP). In start.sh file user changes default base 8080 port to any port not intersecting ports that are already in use. After that related ports are set automatically. Each of them is formed according to following incrementation pattern:

- port for listening to ConnectionRequest = default base 8080 port +2;
- Hosthandler interface port = default base 8080 port +1;
- port for listening to Kicked requests = default base 8080 port +4;
- port for listening to UDP echo diagnostics = default base 8080 port +8;
- file transfer (download/upload) port = default base 8080 port +16.

Please note: Default base 8080 port is used if in start.sh file following line can be found:

```
until ./dimclient
```

For example, if instead of `until ./dimclient` user specifies `until ./dimclient 8090` then port for listening to ConnectionRequest will be 8092, Hosthandler interface port – 8091, port for listening to Kicked requests – 8094, port for listening to UDP echo diagnostics – 8088, file transfer (download/upload) port – 8096. While starting next Client(s) remember those ports are occupied and cannot be used as base port value in start.sh file.

For given example you cannot use any of the following ports as base port for next starting Client: 8080, 8081, 8082, 8084, 8088, 8096.

Please remember that other ports can be occupied for STUN server or other needs. Check data-model.xml file to be sure port you want to specify as next Client's base port is available.

Please note: If UDP Echo port is set in data-model.xml explicitly set port will be used as default instead of one set using incrementation pattern.

16. Client Configuration File

Client configuration file – dimclient.conf has the following fields:

- ManagementServerURL – URL for the CPE to connect to the ACS using the CPE WAN Management Protocol. This parameter must be in the form of a valid HTTP or HTTPS URL. If the parameters is not specified, the value of [InternetGateway]Device.ManagementServer.URL will be taken from the data-model.xml.
- Username – username used to authenticate the CPE when making a connection to the ACS using the CPE WAN Management Protocol. This username is used only for HTTP-based authentication of the CPE. If the parameters is not specified, the value of [InternetGateway]Device.ManagementServer. Username will be taken from the data-model.xml.
- Password – password used to authenticate the CPE when making a connection to the ACS using the CPE WAN Management Protocol. This password is used only for HTTP-based authentication of the CPE. If the parameter is not specified, when read from data-model.xml, this parameter returns an empty string, regardless of the actual value.
- maxHTTPLogSize (in bytes) – size of the HTTP.log. If the parameter is not set or its value is 0, there will be no limits on HTTP.log file size. Thus, the backup log files will not be created and the subsequent maxHTTPLogBackupCount parameter will be ignored.
- maxHTTPLogBackupCount – number of backup log files that will be created and stored for HTTP. log. Each HTTP.log backup log file should be of the size stated in preceding maxHTTPLogSize parameter. Each time the HTTP.log reaches maxHTTPLogSize, the new backup log is created with index 1, while the oldest one is deleted if the total HTTP.log backup count exceeds maxHTTPLogSize. If maxHTTPLogBackupCount is not set or its value is 0, the backup log files will not be created and when the HTTP.log reaches maxHTTPLogSize, it will be purged without creating any backup logs.
- MaxTestLogSize – size of the TEST.log If the parameter is not set or its value is 0, there will be no limits on TEST.log file size. Thus, the backup log files will not be created and the subsequent MaxTestLogBackupCount parameter will be ignored.
- MaxTestLogBackupCount – number of backup log files that will be created and stored for TEST.log. Each TEST.log backup log file should be of the size stated in preceding MaxTestLogSize parameter. Each time the TEST.log reaches MaxTestLogSize, the new backup log is created with index 1, while the oldest one is deleted if the total TEST.log backup count exceeds MaxTestLogSize. If MaxTestLogBackupCount is not set or its value is 0, the backup log files will not be created and when the TEST.log reaches MaxTestLogSize, it will be purged without creating any backup logs.
- HTTPLogFilePathName – path and name of HTTP Log File (for example tmp/HTTP.log). If you don't want to log the information you should use the "/dev/null" filename for this log file.
- TESTLogFilePathName – path and name of TEST Log File (for example tmp/TEST.log). If you don't want to log the information you should use the "/dev/null" filename for this log file.

17. Log Configuration

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

The following levels of logging exist in Dimark Client:

- DEBUG – debug data is logged (logs 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.

Please note: DEBUG level is the highest and logs 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.in.

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>
DU_TRANSFER;INFO <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>
DB;INFO <NL>
DU;INFO <NL>
<EOF>
```

where:

<NL> – new line symbol;

<EOF> – end of line symbol.

18. Command Line Switches for dimclient Application

To view the definition of command line parameters execute `# ./dimclient -h` command.
The following runtime parameters are implemented in Dimark Client:

`-h` – help. Lists all available runtime parameters with short description;

`-d` – IP address for binding Connection Request handler. When there are several network interfaces an interface for Connection Request listening should be specified;

`-b` – base port for binding Connection Request handler;

`-p` – temporary folder for runtime DB (must be writable);

`-i` – proxy host;

`-o` – proxy port;

`-f` – configuration file (ManagementServer.URL, ManagementServer.Username, ManagementServer.Password, etc.). States the location of configuration file;

`-l` – log configuration file path (default “log.config”).

19. Connection to ACS Using HTTP Proxy

If you access the Internet through the proxy server please use the settings listed below:

In start.sh file find the following line and implement needed changes:

```
./dimclient -b8080 -p$CDIR -i<proxy_host> -o<proxy_port>
```

where:

- `-i` – Proxy host;
- `-o` – Proxy port.

20. Client Error Codes

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

Code	Constant	Displayed message	Description
9000	ERR_METHOD_NOT_SUPPORTED	Method not Supported	Method not supported.
9001	ERR_REQUEST_DENIED	Request Denied	Request denied (no reason specified)
9002	ERR_INTERNAL_ERROR	Internal Error	Internal error
9003	ERR_INVALID_ARGUMENT	Invalid Arguments	Invalid arguments
9004	ERR_RESOURCE_EXCEEDED	Resources exceeded	Resources exceeded (when used in association with SetParameterValues, this must not be used to indicate parameters in error)
9005	ERR_INVALID_PARAMETER_NAME	Invalid Parameter Name	Invalid parameter name (associated with Set/GetParameterValues, GetParameterNames, Set/GetParameterAttributes, AddObject, and DeleteObject)
9006	ERR_INVALID_PARAMETER_TYPE	Invalid Parameter Type	Invalid parameter type (associated with SetParameterValues)
9007	ERR_INVALID_PARAMETER_VALUE	Invalid Parameter Value	Invalid parameter value (associated with SetParameterValues)
9008	ERR_READONLY_PARAMETER	Attempt to set a non-writable parameter	Attempt to set a non-writable parameter (associated with SetParameterValues)
9009	ERR_NOTIFICATION_REQ_REJECT	Notification request rejected	Notification request rejected (associated with SetParameterAttributes method)
9010	ERR_DOWNLOAD_FAILURE	Download/ ScheduleDownload failure	File transfer failure (associated with Download, ScheduleDownload, TransferComplete or AutonomousTransferComplete methods)
9011	ERR_UPLOAD_FAILURE	Upload failure	Upload failure (associated with Upload, TransferComplete or AutonomousTransferComplete methods)

Code	Constant	Displayed message	Description
9012	ERR_TRANS_ AUTH_FAILURE	File transfer server authentication failure	File transfer server authentication failure (associated with Upload, Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomoutDUStateChangeComplete methods)
9013	ERR_NO_TRANS_ PROTOCOL	Unsupported protocol for file transfer	Unsupported protocol for file transfer (associated with Upload, Download, ScheduleDownload, DUStateChangeComplete, or AutonomoutDUStateChangeComplete methods)
9014	ERR_JOIN_ MULTICAST_ GROUP	File transfer failure: unable to join multicast group	File transfer failure: unable to join multicast group (associated with Download, TransferComplete or AutonomousTransferComplete methods)
9015	ERR_CONTACT_ FILE_SERVER	File transfer failure: unable to contact file server	File transfer failure: unable to contact file server (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomoutDUStateChangeComplete methods)
9016	ERR_ACCESS_ FILE	File transfer failure: unable to access file	File transfer failure: unable to access file (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomoutDUStateChangeComplete methods)
9017	ERR_COMPLETE_ DOWNLOAD	File transfer failure: unable to complete download	File transfer failure: unable to complete download (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomousDUStateChangeComplete methods)
9018	ERR_CORRUPTED	File transfer failure: file corrupted or otherwise unusable	File transfer failure: file corrupted or otherwise unusable (associated with Download, TransferComplete, AutonomousTransferComplete, DUStateChangeComplete, or AutonomoutDUStateChangeComplete methods)
9019	ERR_FILE_ AUTHENTICATION	File transfer failure: file authentication failure	File transfer failure: file authentication failure (associated with Download, TransferComplete or AutonomousTransferComplete methods)

Code	Constant	Displayed message	Description
9020	ERR_COMPLETE_DOWNLOAD_TIME_WINDOWS	File transfer failure: unable to complete download within specified time windows	File transfer failure: unable to complete download within specified time windows (associated with TransferComplete method)
9021	ERR_CANCELATION_NOT_PERMITTED	Cancelation of file transfer not permitted in current transfer state	Cancelation of file transfer not permitted in current transfer state (associated with CancelTransfer method)
9022	ERR_INVALID_UUID_FORMAT	Invalid UUID format	Invalid UUID Format (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install, Update, and Uninstall)
9023	ERR_UNKNOWN_EXEC_ENVIRONMENT	Unknown Execution Environment	Unknown Execution Environment (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install only)
9024	ERR_DISABLE_EXEC_ENVIRONMENT	Disabled Execution Environment	Disabled Execution Environment (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install, Update, and Uninstall)
9025	ERR_DU_EXEC_ENVIRONMENT_MISHMATCH	Deployment Unit to Execution Environment Mismatch	Deployment Unit to Execution Environment Mistmatch (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install and Update)
9026	ERR_DUPLICATE_DU	Duplicate Deployment Unit	Duplicate Deployment Unit (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install only)
9027	ERR_SYSTEM_RESOURCES_EXCEEDED	System Resources Exceeded	System Resources Exceeded (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install and Update)
9028	ERR_UNKNOWN_DU	Unknown Deployment Unit	Unknown Deployment Unit (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Update and Uninstall)

Code	Constant	Displayed message	Description
9029	ERR_INVALID_DU_STATE	Invalid Deployment Unit State	Invalid Deployment Unit State (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Install, Update and Uninstall)
9030	ERR_INVALID_DU_UPDATE_DOWNGRADE_NOT_PERMITTED	Invalid Deployment Unit Update - Downgrade not permitted	Invalid Deployment Unit Update – Downgrade not permitted (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Update only)
9031	ERR_INVALID_DU_UPDATE_VERSION_NOT_SPECIFIED	Invalid Deployment Unit Update - Version not specified	Invalid Deployment Unit Update – Version not specified (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Update only)
9032	ERR_INVALID_DU_UPDATE_VERSION_ALREADY_EXISTS	Invalid Deployment Unit Update - Version already exists	Invalid Deployment Unit Update – Version already exists (associated with DUStateChangeComplete or AutonomoutDUStateChangeComplete methods: Update only)
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.
9805	ERR_DIM_MARKER_OP		Error during creating or deleting one of markers for detecting boot or bootstrap.
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.
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.

Code	Constant	Displayed message	Description
9823	ERR_WRITE_ OPTION		Unable to write voucher information to persistent storage.
9830	ERR_READ_ PARAMFILE		Error reading the initial parameter file or data from storage or parameter metadata from storage.
9831	ERR_WRITE_ PARAMFILE		Error during writing data into storage or parameter metadata into storage.
9890	ERR_NO_INFORM_ DONE		Error when the Inform was not successful.

References

1. TR-069, CPE WAN Management Protocol (CWMP), Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-069_Amendment-3.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-5.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_Amendment-1.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-2_Amendment-2.pdf.
13. TR-196, Femto Access Point Service Data Model, Broadband Forum Technical Report, http://www.broadband-forum.org/technical/download/TR-196_Amendment-1.pdf.
14. cwmp-datamodel-1-2.xsd, Broadband Forum XML Schema http://www.broadband-forum.org/cwmp/cwmp-datamodel-1-2.xsd
15. Organizationally Unique Identifiers (OUIs), http://standards.ieee.org/faqs/OUI.html
16. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt
17. RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, http://www.ietf.org/rfc/rfc2617.txt

Annex A. TR-069 Client FAQ

Please see http://www.dimark.com/client_release/index.html for the latest information.

General Questions

Q: Client memory consumption has Threads parameter. What does it mean?

A: Dimark Client has two main threads – Main and Host. They cannot be turned off. Other possible threads are:

- Connection Request
- UDP Connection Request
- UDP Echo
- Transfers
- STUN
- kicked handler
- timeHandler
- diagnosticsThread
- passiveNotificationHandler
- activeNotificationHandler

Number reflecting the threads depends on compiler flags that are set.

Q: We will be targeting non-Linux platforms and will need to port to our platform OS, BSD socket interface, and SSL interface. We would like to understand what level of support we can expect from Dimark for this port/ integration effort.

A: Our focus is on deployments and it is in Dimark's best interest to get our partners shipping products ASAP. We support any questions via email on how to bring the client code to the target platform as well on how to integrate the client with the host system.

Q: Based on your experience with your client, what is the estimated port / integration time to a new platform (consider our target is not Linux based).

A: The typical porting for a Linux (or similar POSIX-based OS, such as VxWorks) has been between 1 day and 1 week. If your platform is non Linux based, but provides socket interfaces. The porting work is primarily for SOAP.

Q: Can you provide details of the test and certification services? The proposal states our devices will be fully certified in your Certification Lab. What if we would like additional passes, for example, for a new platform or for an extended data model?

A: The certification is based on a single CPE. Additional fees would arise for additional CPE systems. Multiple passes (within a reasonable limit) until passing the certification of a single CPE are expected.

Q: What are all the system resources required? (threads/tasks, sockets, timers, semaphores, etc.)

A: BSD sockets, threads. The client does not use semaphores or other means. There is another port used for communication with the host system.

Q: Are any tools required to build the sources beyond the basics GCC toolchain?

A: SSL is not part of our distribution of the client code. We provide the hooks for OpenSSL with our client. Additionally we are currently working on Mocana NanoSSL.

Q: Is there any third party copyrighted code provided or required beyond Dimark's? Any additional licenses, sub-licenses we would need? Any open source and/or GPL code? (beyond gSOAP)?

A: Except a commercial license agreement for gSOAP Version 2.8.8 (standard commercial edition) from Genivia (http://www.genivia.com/Products/gsoap/contract.html). Dimark Client requires OpenSSL which can be replaced with Mocana NanoSSL in the near term.

Q: We already have a gSOAP client and understand you can help us modify it for TR-069 compliance. Will these modifications disrupt our existing SOAP protocols?

A: If you already have gSOAP on the system, the majority of porting work for the client has already been done, significantly reducing the porting time for the client to your platform. The modifications for TR-069 will not affect other gSOAP operations. However we require a specific gSOAP version to be used with our client due to the modifications (Version 2.8.8). The modification has to do with the bi-directional operation of TR-069, which is outside of the intended scope of SOAP. gSOAP is used as source code generator for the SOAP operations.

Q: What Linux Version is the client compatible with?

A: The client is compatible with Linux kernel releases 2.4, 2.6 as well as any other POSIX-based operating system.

Q: When Connection Requests are sent to the Client more frequently, the Client returns 503 Error (Service Temporarily Unavailable). What is happening?

A: The client processes no more than 10 Connection Requests in 100 seconds. When Connection Requests are more frequent, the Client returns 503 error. This logic was implemented to prevent DoS attacks.

Debug Questions

Q: The Client cannot be connect to the ACS.

A: This question assumes that Client could reach the ACS, but ACS rejected the Client for some reason. In this case user should browse the Debug log. The most common is 8003 error that is returned if one of the key parameters (manufacturer OUI, Product Class or Serial Number) is stated not incorrectly. Please make sure that Client sends a 6-digit number as manufacturer OUI, that Product Class is stated (unless Dimark ACS version 2.3.2 or higher is used. This versions follow TR-069 Amendment 3 instructions according to which the product class identifier is optional) and that Serial Number is stated.

Q: The Client doesn't "see" the ACS.

A: This situation means that Client couldn't connect to ACS and there is no Debug log. It is a result of one of the following reasons:

- there is no physical connection with the ACS (either the Internet doesn't work or the ACS is down);
- proxy settings on Client side are wrong.

In the first case user may check the physical connection with the help of Telnet command. Open a command line and type the following telnet command:

```
telnet <ACS URL> <port>
```

If ACS is online a test connection will be established.
In the second case user should check Client proxy settings. In start.sh file find the following line and implement needed changes:

```
./dimclient -b8080 -p$CDIR -i<proxy_host> -o<proxy_port>
```

where:

`-i` – Proxy host;
`-o` – Proxy port.

Engineering Questions

Q: What gSoap version do I need?

A: We require using gSOAP version 2.8.8.

Q: When I use "make", "make clean" and then "make", I get an error message. Could Dimark tell me that if it is bug or not?

A: This is a known issue with make. For some obscure reason make does not recognize the creation of soapC.c after a "make clean" is done. Starting make a second time will cure the problem.

Q: I modify the Client configuration (./configure --without-openssl) to make the source code without SSL and I get some error message. Could Dimark tell me that's the problem?

A: The configuration does still include the DIGEST authentication, which in turn requires SSL libraries being present. Removing the plugin/httpda.c and plugin/smdevp.c modules will cure this situation.

Q: Could Dimark give me the sample configuration files for the program to access the testing server, "http://test.dimark.com:8080/login"?

A: Define http://test.dimark.com:8080/login as current InternetGatewayDevice.ManagementServer.URL in the data-model.xml. The client will take the OUI-SNR as Username and Password per TR-069. Our ACS will allow connectivity with those credentials.

*Q: Can I use the *.param datamodel file if the Client vesion is 4.0 or higher?*

A: Please do the following in start.sh to work with *.param:

```
#./conv-util data-model.xml > $CDIR/tmp.param
cp -f dps.param $CDIR/tmp.param
```

Please note: dps. param should be in the same folder as start.sh.

Q: How to switch to cwmp-1-2?

A: Please change the cwmp version in the following strings of `methods.h` file and compile the client:

```
//gsoap cwmp service name: dim
//gsoap cwmp service style: rpc
//gsoap cwmp service encoding: encoded
```

```
//gsoap cwmp service namespace: urn:dslforum-org:cwmp-1-2
//gsoap cwmp schema  namespace: urn:dslforum-org:cwmp-1-2
```

Q: How to redirect screen log to external file?

A: When the start.sh file is run, operator should identify the external file for screen logs. It can be done according to the following example:

```
./start.sh > log.txt
```

In this case the log will be stored in log.txt file.

Q: How should I confirm the performing or applying of the Schedule Download if I choose the "4 Confirmation Needed" WindowMode for the Schedule Download?

A: As the confirmation type depends on the used environment, integrator should modify the `isConfirmationTrue()` function located in `src/ftcallback.c`:

```
/* if Confirmation is true then function returns 1
* else returns 0
* */
int isConfirmationTrue(TransferEntry *te)
{
/*
"The CPE MAY support "4 Confirmation Needed". This means that
the CPE MUST ask for and receive confirmation before performing
and applying the download. It is outside the scope of this
specification how the CPE asks for and receives this confirmation.
If confirmation is not received, this time window MUST NOT be used."
*/
// TODO: ask the confirmation
return false; // default
}
```

SSL Questions

Q: How to build the Client for supporting HTTPS?

A: You need to have `openssl-devel` package installed.
Go to the project root folder and run `configure` file. Possible options are as follows:

`./configure` - Build Client with HTTPS support, but without SSL authentication.
`./configure --with-clientsslauth` - Build Client with client SSL authentication. Server checks client certificate.
`./configure --with-clientsslauth` - Build Client with server SSL authentication. Client checks server certificate.
`./configure --with-clientsslauth --with-clientsslauth` - Build Client with both client and server SSL authentication. Both server and client checks SSL certificate.

`Configure` file automatically creates `Makefile` file in root directory and project is ready to be compiled with the help of `make` command.
If the above actions were performed not on the "fresh" client, perform `make clean` before `make`.
Default settings is not to use SSL certificate authentication, just plain SSL.

Q: Does Dimark uses OpenSSL?

A: Yes, this is the only SSL implementation tested so far.

Q: Is this OpenSSL built as part of Dimark? If not, which version of OpenSSL do we need to use?

A: No, Dimark client uses default OpenSSL version. Currently 0.9.8 and 1.0 versions are supported.

Q: Does dimark use cURL?

A: No, cURL is not used. Dimark uses gSOAP as a HTTP server/client.

Q: What is the version of TLS supported by dimark for HTTPS?

A: TLS support is provided by OpenSSL. Thus, TLS 1.0 is supported with OpenSSL 1.0.

Q: Where do we need to store the Certificates for SSL authentication?

A: Certificate is loaded relative to the directory where dimclient binary is run. To enable loading, you need to modify file src/dimclient.c, function soap_ssl_client_context() to use the certificate.

Sample:

```
soap_ssl_client_context( &soap,
SOAP_SSL_DEFAULT,
NULL, // keyfile, SSL_CTX_use_certificate_chain_file
NULL, // password, SSL_CTX_set_default_passwd_cb
“certificate.pem”, // cafile, SSL_CTX_load_verify_locations
NULL, // capath, SSL_CTX_load_verify_locations
NULL // randfile
)
```

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.

Annex B. Readme Files

Annex B stores files that previously were located in README folder of the Client.

README

README – main README file containing information of how parameters are handled and how functionality works. Usage of HTTP BasicAuthentication and default parameter file structure are also described in this file.

How Parameterhandling works:

Parameters are one of the key elements in the TR-069. Therefor is a lot of work done in the ParameterHandling part.

Storage of parameter in memory.

The parameters have a hierarchical structure. In the client the parameters are also stored in an tree where the parents are the nodes and the children are the leaves.

Example: For InternetGatewayDevice.DeviceInfo.Manufacturer parameter:

- InternetGatewayDevice. is the parent of the entry DeviceInfo;
- DeviceInfo. is the parent of Manufacturer.

Every parameter is stored in an object of the structure ParameterEntry (see parameter.c) which has 4 pointers to handle the tree.

- child – is a pointer to the first child of this entry;
- next – is a pointer to the next sister of this entry (NULL means there is none);
- prev – is a pointer to the previous server of this entry (NULL means there is none);
- parent – is a pointer to the parent (node) of this entry.

These pointers have access to all entry information.

The exception is rootEntry which is for management reason. It owns the children of the highest parameter level.

Example with !! NULL pointers are left off for easier reading !!

```
        InternetGatewayDevice.DeviceInfo.Manufacturer
        InternetGatewayDevice.DeviceInfo.ModelName
  rootEntry
        .parent = NULL
        .next   = NULL
        .prev   = NULL
        .child    -> InternetGatewayDevice.
                              .parent ->  rootEntry
```

```
.child  ->      DeviceInfo.
                    .parent -> InternetGatewayDevice.
                    .child  -> Manufacturer
                                    .parent -> DeviceInfo.
                                    .next -> ModelName
                                                .parent -> DeviceInfo.
                                                .prev   -> Manufacturer
```

Special handling is made for the MultiObjectTypes, which are nodes that can have more then one instance. The instances are numbered from 1 to n. The instance is handled a node. The numbering is made in the parent node of the instance.

Parameter value or default value

Action	read only	readWrite	value	default value
setParameterValue	-	x		
getParameterValue				
writeToFile				
readFromFile				
AddObject	x	x		x

Parameter persistence directory

If dimclient is compiled without SQLite DB support, parameters are stored in files in a flash file system. The path to the directory is defined in dimclient.c as PERSISTENT_PARAMETER_DIR. For every parameter is a single file with the parameter path as filename. The internal structure is the same as the initial setup file. The directory must exist.

File name describing the parameters

To build the tree and preset the parameter with values, there is a initialization file. The file pathname is defined in dimclient.c as DEFAULT_PARAMETER_FILE. This file can be in a read only file system, for example ./etc/config/tmp.param

Parameter access to host system

To inform the host system about a changed parameter or to get a parameter value from the host system, there are access functions defined. In the parameter structure are 3 indexes into arrays of access functions. These arrays are defined in paramaccess.c.

These indexes are:

- int initDataIdx index to the init and delete function
- int getDataIdx index to the get function,

- int setDataIdx; index to the set function

Two special values are defined for the index:
-1 function not allowed
0 get or set value from the parameter, no call to the host system

The index must be a unique integer.
The access to the host system is done by two convenient functions.
int getAccess(int idx, const char *name, ParameterType type, ParameterValue *value);
int setAccess(int idx, const char *name, ParameterType type, ParameterValue *value);

The function prototype is defined as
typedef int (*accessParam)(const char* , ParameterType, ParameterValue *);

Example:

Parameter: InternetGatewayDevice.WANDevice.1.WANCommonInterfaceConfig.TotalBytesSent
.initDataIdx = -1
.getDataIdx = 817
.setDataIdx = 817

There are two access functions defined by the host system, which are set in the arrays

```
Func getArray[] = {
{ 817 , &getETH0SentBytes }
};

Func setArray[] = {
{ 817 , &setETH0SentBytes }
};
```

Callback Feature
To execute a function after setting a value, the access function are used. But if you want to execute a function after the communication is done, the callback has to be used.
The callback feature gives you the possibility to register a callback function which is called after the communication with the server has finished.
The callback functions are stored in a list are executed as a FIFO (first in first out).
The callback function is only called once, and there is no persistence of the callback list.

There is a function to register a callback:
void addCallback(Callback cb, List* list);

where:
cb – points callback function,
list – points callback list from the following:

- List initParametersDoneCbList;
- List preSessionCbList;
- List postSessionCbList;
- List cleanUpCbList;

The Callback prototype
typedef int (*Callback) (void);

For example see: callback.c file

Using HTTP BasicAuthentication
Username and password are taken from Parameter:

InternetGatewayDevice.ManagementServer.Username
InternetGatewayDevice.ManagementServer.Password

These parameter can be changed by the server.
The username is the OUI which are the 24 bits of the MACAddress a hyphen and the serial number.
The password is the the serial number, a hyphen and the OUI.

Appendix A:
This to do for Parameter Access:

1. Implement the Access to the Operating System
2. Declare the Functions in an Header file
3. Include the header file in paramaccess.c
4. Add the getFunctions to the getArray[] (Caution!! use unique numbers)
5. Add the setFunctions to the setArray[]
6. Create an entry in the Default parameter file. The filename is defined in parameter.c. The file can be in a RO file system.
7. After the change of writable parameter is a file written with the value of the parameter. This filename is equal the parameter name. The directory name is defined in parameter.c

This directory must live in an RW file system, which is persistent.
These parameter files are read after the load of the default parameter file, and overwrites the default value of the parameter.

END OF SECTION

Default Parameter file structure:

notification = notify the server if the value of this parameter is changed through the LAN interface
0 = NotificationNone
1 = NotificationPassive (included in the next regular inform schedule)
2 = NotificationActive (changes of this parameter initiate an immediate inform schedule)
4 = NotificationAllways (always included in the inform schedule)

notificationMax = maximum allowed notification type
0 = NotificationMaxNone
1 = NotificationMaxPassive (included in the next regular imform schedule)
2 = NotificationMaxActive (changes of this parameter initiate an immediate inform schedule)
3 = NotificationMaxAllways (always included in the inform schedule)
4 = NotificationNotChangeable (changes are not allowed)

reboot = Reboot the system if the value of this parameter has changed
0 = Reboot
1 = NoReboot

initIdx = -1 , not used yet, value is taken from default value
getIdx = index in the getArray[] or
-1 if no support for the get() means WriteOnly
0 get the Value from the memory
1 get the Value from the persistent storage
n get the Value from a customer specific implementation

setIdx = index into the setArray[] or
-1 if no support for the set() means ReadOnly
0 store the Value in memory, lost after a reboot
1 store the Value in the persistent storage
n store the Value in a customer specific implementation

!!setIdx and getIdx should always have the same value, you could not mix 0 and 1 values for one param!!

access list = collection of the access listen tries, delimited by '|'

default value = setup value, must be casted to void *

Debug

For environments where there is no debugging functionality available there is the option to enable output of the client to help diagnose issues.
Those are defined in debug.h and implemented in debug.c

To minimize performance penalties during debugging output the debugging output can be removed completely from the program code. This is managed through the WITHOUT_DEBUG=TRUE in Makefile.

Each line in log.config has the next format:

subsystem;level

[SOAP|MAIN|PARAMETER|TRANSFER|STUN|ACCESS|MEMORY|EVENTCODE|SCHEDULE|ACS|VOUCHERS|OPTIONS|
DIAGNOSTIC|VOIP|KICK|CALLBACK|HOST|REQUEST|DEBUG|AUTH];[DEBUG|INFO|WARN|ERROR]

For example:

SOAP;DEBUG
MAIN;INFO
PARAMETER;WARN
TRANSFER;ERROR
STUN;DEBUG
ACCESS;DEBUG
MEMORY;DEBUG
EVENTCODE;DEBUG
SCHEDULE;DEBUG
ACS;DEBUG
VOUCHERS;DEBUG
OPTIONS;DEBUG
DIAGNOSTIC;DEBUG
VOIP;DEBUG
KICK;DEBUG
CALLBACK;DEBUG
HOST;DEBUG
REQUEST;DEBUG
DEBUG;DEBUG
AUTH;DEBUG

Functionality: Download

Description:
Downloads one file per ACS request from a known server.
The file is specified by an URL.
ACS can specify an delay of the download function.
A delay value > 0 means the download should be started after a stated delay value (in seconds).
Before download and after successful download, a callback functions are called.
The callback functions are registered in dimclient.c and must be set at the startup of dimclient.
If the value of the callbackfunction is NULL, no callback is initiated.
See ftcallback.c for an example.

The function declaration of the Callback function:

```
/* Callback function is called before a file is downloaded */
int fileDownloadCallbackBefore(TransferEntry * te);

/* Callback function is called after a file is downloaded */
int fileDownloadCallbackAfter(TransferEntry * te)
```

Return code should be one of error codes defined in dimark_globals.h or 0 in case of successful completion.

Example of a callback function implementation:

```
/* Callback function is called before a file is downloaded */
int fileDownloadCallbackBefore(TransferEntry * te)
{
int ret = OK;

DEBUG_OUTPUT (
dbglog( SVR_DEBUG, DBG_TRANSFER, "Download Callback Before: %s
Filetype: %s\n", te->targetFileName, te->fileType);
)

return ret;
}
```

Functionality: Uploads

Description:
This method MAY be used by the ACS to cause the CPE to upload a specified file to the designated location.
The file is specified by an URL.
ACS can specify an delay of the upload function.
A delay value > 0 means the upload should be started after a stated delay value (in seconds).
The callback functions are registered in dimclient.c and must be set at the startup of dimclient.
If the value of the callbackfunction is NULL, no callback is initiated.
See ftcallback.c for an example.

The function declaration of the Callback function:

```
/* Callback function is called before a file is uploaded */
int fileUploadCallbackBefore(TransferEntry *te);

/* Callback function is called after a file is uploaded */
int fileUploadCallbackAfter(TransferEntry *te);
```

Return code should be one of error codes defined in dimark_globals.h or 0 in case of successful completion.

Example of a callback function implementation:

```
/* Callback function is called after a file is uploaded */
int fileUploadCallbackAfter(TransferEntry *te)
{
int ret = OK;

DEBUG_OUTPUT (
dbglog( SVR_DEBUG, DBG_PARAMETER, "Upload Callback After:
targetFileName = %s  Checksum = %d\n", te->targetFileName, 0 );
)

return ret;
}
```

Functionality: Schedule Downloads

Description:
This method may be used by the ACS to cause the CPE to download a specified file from the designated location and apply it within either one or two specified time windows.
The file is specified by an URL.
After a successful download, a callback function is called.
The callback functions are registered in dimclient.c and must be set at the startup of dimclient.
If the value of the callback function is NULL, no callback is initiated.
See ftcallback.c for an example.

The function declaration of the Callback function:

```
/* Callback function is called before a file is downloaded */
int fileScheduleDownloadCallbackBefore(TransferEntry *te, int
isFirstTimeWindow);

/* Callback function is called after a file is ScheduleDownloaded */
int fileScheduleDownloadCallbackAfter(TransferEntry *te, int
isFirstTimeWindow)
```

Return code should be one of error codes defined in dimark_globals.h or 0 in case of successful completion.

Example of a callback function implementation:

```
/* Callback function is called before a file is downloaded */
int fileScheduleDownloadCallbackBefore(TransferEntry *te, int
isFirstTimeWindow)
{
int ret = OK;

DEBUG_OUTPUT (
dbglog( SVR_DEBUG, DBG_TRANSFER, "ScheduleDownload Callback Before:
%s  Filetype: %s\n", te->targetFileName, te->fileType);
)
return ret;
}
```

Functionality: Client Notification

Description:
To read and write the client parameters by the host system (not the ACS), there is a thread spawned in the dimclient.
The thread is listening at port 8081 for a command to get or to set a parameter.

GET:

send	get	
send	<parameterpath>	
receive	<type>	(see parameter.h)
receive	<value>	all values are returned as string
		in case of an error type and value has same value
receive	<emptyline>	

SET:

send	set	
send	<parameterpath>	
send	<parametervalue>	
receive	<returncode>	
receive	<emptyline>	

ADDOBJECT:

send	add	
send	<objectpath>	
receive	<instanceno>	instance number of new object
		in case of an error a 0 is returned
receive	<emptyline>	

DELOBJECT:

send	del	
send	<objectpath>	
receive	<errorcode>	0 = noError
receive	<emptyline>	

CLEAROPTION:

send	opt	
send	<voucherSN>	
send	<remove>	
receive	<errorcode>	0 = noError

receive <emptyline>

GETVENDORINFO

send vg
receive <errorcode> 0 = noError
receive <manufactorerOUI>
receive <serialNumber>
receive <productClass>
receive <emptyline>

SETVENDORINFO

send vs
send <manufactorerOUI>
send <serialNumber>
send <productClass>
receive <errorcode> 0 = noError
receive <emptyline>

ADDDEVICEINFO

send da
send <manufactorerOUI>
send <serialNumber>
send <productClass>
receive <errorcode> 0 = noError
receive <emptyline>

REMOVEDEVICEINFO

send dr
send <manufactorerOUI>
send <serialNumber>
send <productClass>
receive <errorcode> 0 = noError
receive <emptyline>

DHCPDISCOVERY

send hd
send <manufactorerOUI>
send <serialNumber>
send <productClass>
receive ho
receive <manufactorerOUI>

receive <serialNumber>
receive <productClass>
receive <URL>
receive <emptyline>

DHCPREQUEST
send gr
send <manufactorerOUI>
send <serialNumber>
send <productClass>
receive ho
receive <manufactorerOUI>
receive <serialNumber>
receive <productClass>

For testing purposes the access is not delimited to the localhost.

Functionality: Server Notification

Description:
The ACS can notify the CPE to initiate a inform message.
The Communication is started by a HTTP Request from the ACS to port 8082 of the CPE.
The used authentication is Digest. The following Parameters are checked before a connection is allowed:
InternetGatewayDevice.ManagementServer.ConnectionRequestURL must be identical with the HTTP Request
InternetGatewayDevice.ManagementServer.ConnectionRequestUsername must be the same used by the ACS
InternetGatewayDevice.ManagementServer.ConnectionRequestPassword must be the same used by the ACS

The Connection timing is delimited to one connection every 60 seconds (not implemented yet)
After the connection is approved the socket is closed and an inform Message is sent to ACS.
If the CPE is in an transaction with the ACE the call is ignored.

Communication keep-alive

Jboss Tomcat has maxKeepAliveRequests Variable in the deploys/jbossweb-tomcat50.sar/server.xml file.
This parameter must be set to a value > 100 default = 20
The parameter is found in <Connector .. maxKeepAliveRequests=”1000” .. >

If the value is not set, the connection is closed after x Requests from the client.
Don’t forget to restart jboss

Digest Authentication

To use the Digest authentication, gSOAP Version 2.8.8 and the httpda.* plugin is necessary.
The first inform call is made with Basic authentication, if this fails with return Code 401 the Digest authentication is used. All following calls are made with the setup of this authentication. The next inform call starts with Basic authentication again.

Voip Abstraction Layer

tbd.

TR-111

Device:

Requirements:
Dimclient must be started before the DHCP client starts.
The mainloop in dimclient must wait until the hostsystem has finished the DHCP Session.

Procedure:
Get the device vendor specific informations (ManufacturerOUI, SerialNumber, ProductClass) from dimclient. Use HostInterface with a convenient access function.

- Device.Info.ManufacturerOUI
- Device.Info.SerialNumber
- Device.Info.ProductClass

DHCP Client uses the information for getting the GatewayDevice informations

Store the GatewayDevice vendor specific informations in

- Device.GatewayInfo.ManufacturerOUI
- Device.GatewayInfo.SerialNumber
- Device.GatewayInfo.ProductClass

the mainloop is unlocked, and the first inform message is sent to the ACS.

The DHCP Client uses the host interface to set the parameters if the lease is released or expires without renewal to empty values or to the new values.

Gateway:

Requirements:
Dimclient must run before the first request can be handled by the DHCP server.
The mainloop must not wait for the hostsystem is finished

The DHCP Server reads the data at the first clientrequest.

The DHCP Server stores the client data via hostinterface in dimclient. The server must deliver all 3 values, even one of them is empty.

If a new client dataset is coming from the hostinterface, the data is added or replaces the old one.
The clientdataset is stored in

- InternetGatewayDevice.ManagementServer.ManageableDevice.{i}.ManufacturerOUI
- InternetGatewayDevice.ManagementServer.ManageableDevice.{i}.SerialNumber
- InternetGatewayDevice.ManagementServer.ManageableDevice.{i}.ProductClass

The InternetGatewayDevice.ManagementServer.ManageableDeviceNumberOfEntries parameter is updated every time a new ManageableDevice is created or deleted.
Every time the ManageableDeviceNumberOfEntries changes and the last change is inside the ManageableDevicNotificationLimit time, there is an Informmessage to the ACS.

If the Lease of a client is released the dataset is removed from the ManageableDevice table and the ManageableDeviceNumberOfEntries is decreased by one.

Dataflow:

1. The DHCP Server sends the ClientData line by line (OUI, SerialNumber, ProductClass) to the hostHandler with Cmd “da”
2. HostHandler buffers the ClientData

Stun Client Requirements from TR111

1. Determine Public IP address and port for UDP Connection Requests listener
   primary source port UDPConnectionRequests are awaited
   secondary source port Used for binding timeout recovery

   Stun is enabled by STUNEnable Parameter. Extention in parameter.c to read the parameter STUNEnable, STUNServerAddress, STUNServerPort (uint)

STUNMaximumKeepAlivePeriod (uint), STUNMinimumKeepAlivePeriod (uint)
STUNUsername, STUNPassword, NATDetected (= false wenn STUNEnable false)
UDPConnectionRequestAddressNotificationLimit (uint)
UDPConnectionRequestAddress (String)

If no StunServerAddress use the ACS Serveraddress

Binding requests send from my IP Address and Portno defined for UDPConnectionRequests
ACS_UDP_NOTIFICATION_PORT

Addressmapping
Stun Attribute MAPPED-ADRESS: public IP and Port

If STUNUsername and STUNPassword is given and Returncode is 401
Use USERNAME and MESSAGE-INTEGRITY must be sent and received. Otherwise the Values are not valid.
(Send only MESSAGE-INTEGRITY if Stunserver Response is 401)

No support for CHANGE-REQUEST, CHANGED-ADDRESS, SOURCE-ADDRESS, REFLECTED-FROM Attributes and no SharedSecret exchange.

If local IP Address changes (DHCP) the binding procedure must be repeated.

2. Discover the NAT binding timeout and send STUN Bindings requests to keep alive binding
   2 Parameters STUNMinimumKeepAlivePeriod and STUNMaximumKeepAlivePeriod control the timeout for the Keepalive Binding requests. If the Values are not equal, discover the longest keepalive period.

   2 methods to tell the ACS about the binding address.
   First, add CONNECTION-REQUEST-BINDING Attribute into every StunRequest with my IP and Portno.
   if StunPassword is not empty send it in USERNAME Attribute.
   do not use it in the keepalive tests.
3. Handle the UDPConnectionRequestAddress Parameter if binding changes.
4. Listen for UDP Connection Requests messages

README_compile.txt

README_compile.txt – instructions for compiling Dimark Client on Fedora/Centos/RedHat Linux platform.
QUICK-START INSTRUCTIONS FOR COMPILING DIMARK TR-069 CLIENT ON A Fedora/Centos/RedHat LINUX PLATFORM:

1. Unpack the Dimark client source into its own directory.
2. Unpack gsoap_2.8.8.tar.gz into a separate directory and follow the instructions for running config(ure) and compiling on the Linux HOST platform.
   When completed, copy the binary gsoap_2.8.8/soapcpp2 to the gsoap directory (.e.g <dimark_client_src>/gsoap/> under the Dimark client source directory.
3. These instructions assume OpenSSL is already installed on your platform. If not, obtain any version of OpenSSL from 0.9.7f through 0.9.8i and follow the directions for compiling and installing.
4. Cd to the Dimark client source directory. Review the file:
   Makefile
   and edit as required. When ready to compile, type:

```
make clean
make
```

The Dimark client will first call the soapcpp2 process to compile the gSoap directives to produce the necessary files to continue compilation. Some warning messages will be produced by soapcpp2. These are normal and can be ignored.

The compilation will continue.

When complete the following binary will be present:

dimclient

Please note: See the start.sh script for how the client needs to be started and its target data directory prepared.

Start start.sh

README_sslauth.txt

README_sslauth.txt – Dimark Client SSL authentication notes.
DIMARK CLIENT SSL AUTHENTICATION NOTES - OpenSSL Version Only

The Dimark client can be compiled to enable the checking of SSL certificates.

Please note: The TCP Connection Request (ACS -> client) mechanism does not use SSL.
The contents of this document only refer to main (client -> ACS) communicates when that connection is established using SSL (e.g. https://....).

The compilation flag is "WITH_SSLAUTH". Default does not include support.

WHEN NOT INCLUDED, the SSL support simply checks the server's SSL certificate to make sure that it is a valid certificate. Valid certificate checks are very simple and straightforward:

a) Is it a valid SSL certificate and is it signed.

Please note: Self-signed certificates are also accepted, do virtually anyone may put in place an ACS with a "valid" certificate, and

b) The date on the client is between the "Issued On" and "Expires On" dates of
the certificate.

Please note: The most common reason for SSL connection failure is that the date/time on the client is not set correctly. Use a battery backed real-time clock or an NTP (Network Time Protocol) client on the TR-069 client.

WHEN INCLUDED, there are two main options that can be used to more thoroughly validate certificates:

- Client validates Server's certificate
- Server validates Client's certificate(s)

gSoap provides the soap_ssl_client_context() to setup up validation credentials for the normal main ACS conversations. The arguments to the call are:

soap_ssl_client_context(&soap, SOAP_SSL_DEFAULT, keyfile, password, cafile, capath, randfile);

This call is coded (located) in the src/dimclient.c file.

RECOMMENDATION: Unless the customer/integrator has more detailed knowledge of OpenSSL operation and coding (or other cryptographic requirements) the use of Method A) below using the "cafile" option only, is the preferred implementation approach for the client to validate the server's SSL credentials.

CLIENT VALIDATES SERVER'S CERTIFICATE

With this method, the client has one or more "Root Certificate Authority", also called "Root CA" certificates. The idea is that the server's SSL certificate is signed by a known authority, such as Verisign, Thawte, etc. Each authority has their own Root Certificate used to sign server certificates. These root certificates are made publicly available and can be used by clients to verify that a server's certificate was indeed signed officially by that authority. For information on the trusted hierarchy created by this system, see:
http://en.wikipedia.org/wiki/Root_certificate
http://www.tech-faq.com/root-certificate.shtml

To actually download root certificates, there are several sources on the web for this. One is at Verisign:
https://www.verisign.com/support/roots.html

You will need to fill out information and then will obtain the latest root certificates from Verign, Thawte, and GeoTrust.

Also see the OpenSSL SSL_CTX_load_verify_locations() call page at:
http://www.openssl.org/docs/ssl/SSL_CTX_load_verify_locations.html

In order for this method to be successful, the client MUST have a local copy of the Root CA for the server's certificate; e.g. if the server's certificate is signed by a Verisign Root CA certificate, the client must have that certificate available locally. If the server's certificate is privately signed, then the client must have a copy of the private root ca used to sign the server's certificate.

Please note: Dimark self-signs the SSL server certificate for test.dimark.com.
We provide a copy of the Dimark Root CA certificate for testing. It is available as "dimark.pem" in the ./test directory of this release.

This methods also means that if the TR-069 client is expected to verify one of many ACS's, then the client might need to have many of the common root server certificates pre-loaded, or it must be able to Have a root ca locally installed.

There are two ways to specify the location of root ca certificates using the above call. First is to specify a single file "cafile" that contains one or more certs, or a "capath" that points to a directory that contains one or more certificates.

The "cafile" methods is the most common. The "cafile" is a file name of a single file containing one or more PEM encoded root ca certificates.
A PEM (privacy enhanced mail) format certificate is the most common and refers to a Base-64 encoded X.509 certifiate.
See http://en.wikipedia.org/wiki/X.509 for more information.

Please note: “cafile” could be coded as a string variable that is used to point to one of several files, each file containing one or more certificates.
The coding of the call is up to the integrator of the Dimark client.

The “capath” method is provided by OpenSSL that allows a directory to be used to store a single certificate per file. The details on setting up the directory are in the OpenSSL man page reference earlier.

Please note that “cafile” and “capath” may both be used, OpenSSL searches “cafile” first.

DIMARK supplies it’s private root ca in file “dimark.pem”. If the file was located in the same directory as the diclient binary, then a possible coding of the call would be:

```
soap_ssl_client_context( &soap,
                         SOAP_SSL_DEFAULT,
                            NULL,  // keyfile, SSL_CTX_use_certificate_chain_file
                            NULL,  // password, SSL_CTX_set_default_passwd_cb
                            “dimark.pem”,  // cafile, SSL_CTX_load_verify_locations
                            NULL,  // capath, SSL_CTX_load_verify_locations
                            NULL   // randfile
                       )
```

Please note: gSoap is coded to expect the following matching condition: the text host name in the URL address (endpoint address) must match the name of server in the certificate: Therefore, the URL must be coded as:
https://test.dimark.com:8443/dps/TR069
https://test.dimark.com:8443/dps-basic/TR069
https://test.dimark.com:8443/dps-digest/TR069

because “test.dimark.com” is encoded in the server SSL certificate. The IP address may not be substituted or else gSoap will reject the connection.

SERVER VALIDATES CLIENT’S CERTIFICATE(S)

This is a little used feature. It allows the client to have it’s own set of certificates, much like the server SSL certificate. The client’s certificates are signed and all levels of certificates in the chain must be present as required to permit the server to validate the client’s certificate. More information can be found at:
http://www.openssl.org/docs/ssl/SSL_CTX_use_certificate.html
https://www.openssl.org/docs/ssl/SSL_CTX_set_default_passwd_cb.html

Please note: This is a very complicated setup to administer as the certificates must be unique per device. It is not used in practice.

RANDFILE

gSoap allows the SSL function RAND_load_file() to be called using the "randfile" name to allow using a different random number seed startup process. For more information, see:
https://www.openssl.org/docs/crypto/rand.html
https://www.openssl.org/docs/crypto/RAND_load_file.html

gSoap essentially calls the function with the file name and "-1" as the max_bytes argument value: e.g. RAND_load_file(randfile, -1);

In practice and the nature of TR-069, using "randfile" is not used.

README_tr111.txt

The files in this directory implement TR-111 support for the Dimark TR-069 Client.

1. STATUS OF IMPLEMENTATION
TR-111 is divided into two parts:

PART 1 – DHCP Client / Server
Implemented

PART 2 – STUN (RFC3489)
I. Binding Requests over UDP
All TR-111 Part 2 functionality has been implemented except as noted below for Server and Client.

II. Shared Secret Requests
Server: Optional and not implemented
Client: Not implemented.

III. DNS SRV Record Support
Client: not implemented
Please note: STUN server connection information is provided via TR-069 / TR-111 parameters as set by default in the client or as specified by the ACS.

IV. MESSAGE-INTEGRITY
Server: optional, not implemented.
Client: not implemented.

V. Discovery of NAT binding timeout
Server and Client: not implemented.

VI. CONNECTION-REQUEST-BINDING
Server and Client: not implemented.

VII. UDPConnectionRequestAddressNotificationLimit
Client: not implemented. Parameter accepted, but value current not used.

VIII. STUNMaximumKeepAlivePeriod
Client: not implemented. Parameter accepted, but value current not used.

IX. Server secondary source IP address
Server: optional, not implemented.

X. TR-111 Section 2.2.2.2.1 STUN-based Approach
Server: not implemented. System uses Section 2.2.2.2.2
Notification-based Approach only.

Please note last bullet in Section 2.2.2.2.2, server always attempts TCP-based connection requests first.

XI. Under 2.2.1.4 UDP Connection Requests
The STUN implementation has two phases if Stun is enabled with the ManagementServer.STUNEnable parameter: the client exchanges STUN binding information with the STUN server, and then listens for incoming UDP Connection Requests.

If STUN is not enabled with this parameter, the client only listens for incoming UDP Connection Requests.

Overview of Details:

Exchange binding information with STUN server

STUN Client/Server protocol as per RFC 3489. One iteration with timeout, sufficient to client to obtain mapped address information.

Please note: The STUN procedure does not recognize an incoming TR-111 UDP Connection Request.

Listen for incoming UDP Connection requests

Client enters a loop with a UDP recvfrom() call with 10 second timeouts. Loop continues until the minimum time interval in STUNMinimumKeepAlivePeriod is met. Then client tests to see if STUN is enabled and repeats.

METHOD OF OPERATION

PART 1 - DHCP Client / Server
Implemented

PART 2 - STUN (RFC3489)
The STUN client is available in either a TR-098 (Internet Gateway) or TR-106 (Device) configuration. The data model parameter prefix "<pre>" for TR-098 is: "InternetGatewayDevice" and for TR-106: "Device"

The selection of either is set at compile time.

STUN parameters sent by the ACS are:
<pre>.ManagementServer.
" STUNEnable
" STUNServerAddress
" STUNServerPort
" STUNUsername
" STUNPassword
" STUNMaximumKeepAlivePeriod
" STUNMinimumKeepAlivePeriod
" UDPConnectionRequestAddressNotificationLimit

STUN parameters sent by the client using Inform:
<pre>.ManagementServer.
" NATDetected
" UDPConnectionRequestAddress

See Table 6 (page 26) of TR-111 for a full description of these parameters.

See the format of the actual parameters in the data-model.xml file for use by the Dimark client.

Please note: The following parameters are currently ignored by this version of the STUN client:
STUNMaximumKeepAlivePeriod
UDPConnectionRequestAddressNotificationLimit

Please note: The internal default setting is 100 seconds for STUNMinimumKeepAlivePeriod
Setting the value of this parameter greater than 0 will override the default with the specified value.

METHOD OF OPERATION - SERVER

Simply, Dimark provides a stand-alone STUN server that runs in parallel to the ACS on the same server. The STUN server accepts BINDING Requests and issues BINDING Responses as per RFC3478.

In addition, the STUN server has been augmented to act as a relay for ACS UDP Connection Requests. There are two types of Connection Requests that may be issued by the ACS: TCP and UDP. The ACS will always attempt a TCP connection request first. If a non-response is experienced, the ACS will see if <pre>. UDPConnectionRequestAddress exists for the device and that it has a non-null value. If non-null, the ACS will construct a UDP Connection Request and pass it to the STUN server on "localhost" using UDP port 16000. The STUN server will then relay the UDP packet to the client device. As per TR-111, the STUN server will send multiple copies of the UDP connection request. Default is 2 copies.

Please note: In this method, there is no communication from the STUN server to the ACS. All UDP address information is obtained from the client. This is the TR-111 Section 2.2.2.2.2 Notification-based Approach.

METHOD OF OPERATION - CLIENT

The TR-111 STUN client support replaces previous UDP connection request support in the client. The compiler directive "WITH_UDP" has been deprecated and now "WITH_STUN_CLIENT" is used to include compile time support.

On startup, the Dimark client creates a process thread for "stunHandler".

Please note: This procedure is in the main dimclient.c file, otherwise all TR-111 support is under the "tr111/" directory.

The stunHander() process is responsible for:

1. Examining STUNEnable, as set by the ACS.
   If STUN Enabled:
2. Calling the stun_client procedure for exchanging BINDING information with the STUN server.
3. Examining results and updating Inform parameters for notification if the information has recently changed.
   If STUN Disabled:
4. Updating Inform parameters with local connection information
5. Calling the UDP Connection Request handling process for a period of time. If a validated UDP connection request is received, scheduling a signaled Inform.
   If either Enabled or Disabled, then
6. Sleeping, and returning to 1.