DASH SDK 1.1.6

API Reference Manual

February 2010

Version History

REVISION CHART			
Version	**Author(s)**	**Description of Version**	**Date Completed**
0.0.9	Gowtham Shanmukam	Initial Release	February 26, 2008
0.1.0	Gowtham Shanmukam	Added examples for low level API.	March 3, 2008
1.0.0	Gowtham Shanmukam	Added the following 1. Modified the sample code, so the code will compile and user can copy and use the example as it is. 2. Sample output added to few sample codes 3. Added client-server interaction diagram 4. Added design details 5. Modified few write ups	April 2,2008
1.1.0	Manickavasakam Karpagavinayagam	Added Dash 1.1 profiles .	July 17, 2008
1.1.1	Manickavasakam Karpagavinayagam	Added Indication profile,Ethernet profile and Registered profile	September12,2008
1.1.2	Manickavasakam Karpagavinayagam	Document updated to include all mandatory DASH elements	October 16,2008
1.1.3	Manickavasakam Karpagavinayagam	Note Added for broadcast address during the discovery	November 10,2008
1.1.4	Manickavasakam Karpagavinayagam	Added few changes related to Filtercollection & Indicationsubscription	December 16,2008
1.1.5	Manickavasakam Karpagavinayagam	Changes made in indication unsubscribe and renew	January 08,2009
1.1.6	Vinu Vaghasia	Added Capabilities API for following profiles: 1. PowerManagement 2. Battery 3. Boot control 4. BIOS Management 5. Fan 6. RoleBasedAuthorization 7. SimpleIdentityManagement 8. TextRedirection 9. USBRedirection	February 05,2010

		Replaced CLI help out put screen image to reflect addition of capabilities and lots of formatting, text alignment, bulletpointing and cosmetic changes. Update Index to reflect addition of new pages.	

Table of Contents

1 Introduction

The DASH SDK provides software in source and binary form to allow management consoles and management applications to support the DMTF DASH interface. It consists of libraries, command-line (CLI) utilities, and example code that are used to facilitate DASH integration with industry management consoles. The SDK software will be integrated with existing management consoles. code that may be used to integrate DASH into existing and new management applications. A possible use of the SDK is to reduce the time it takes to add DASH capability to existing management consoles. It executes on a remote client and is used to discover/communicate with DASH-enabled management controllers that are integrated with the client PCs being managed by the console. Applications built using the SDK can discover and interact with any DASH-enabled Management Access Points (MAPs) to which hey have network access and the required credentials.

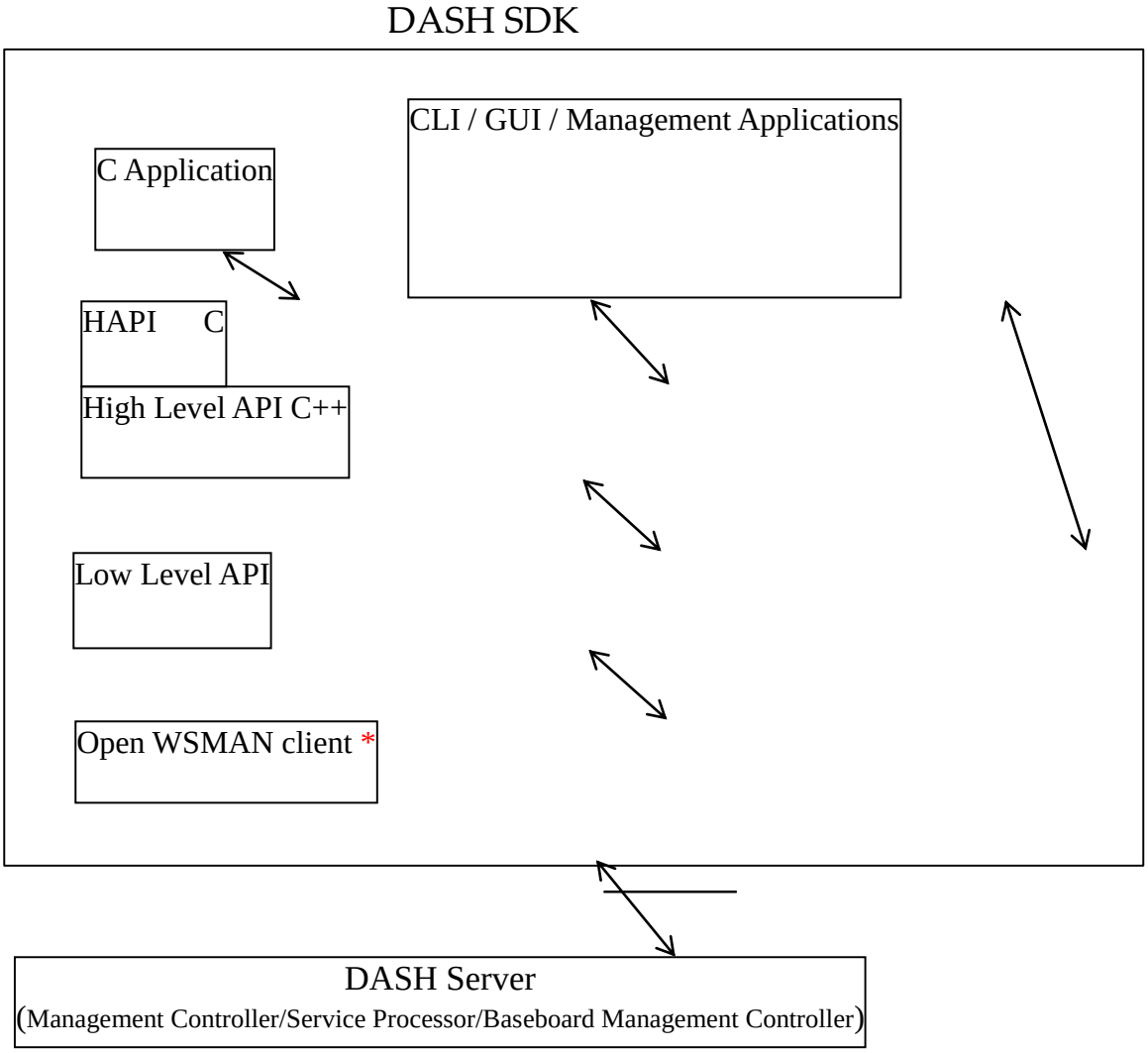
1.1 Acronyms and Abbreviations

This section details any acronyms or abbreviations used in this document

API	Application Programming Interface
BMC	Baseboard Management Controller
CIM	Common Information Model
CIMOM	CIM Object Manager
CLI	Command Line Interface
CLP	Command Line Protocol
CMPI	Common Manageability Programming Interface
DASH	Desktop and Mobile Architecture for System Hardware
DMTF	Distributed Management Task Force,Inc.
DSDK	DASH SDK
HAPI	High Level API
IPMI	Intelligent Platform Management Interface
KVM	Keyboard, Video, and Mouse
MC	Management Controller
MOF	Managed Object Format
SP	Service Processor
WBEM	Web Based Enterprise Management
WSMAN	Web Service for Management

1.2 DASH Client SDK – DASH Server Interaction

This section shows the DASH client SDK interaction with Dash Server.

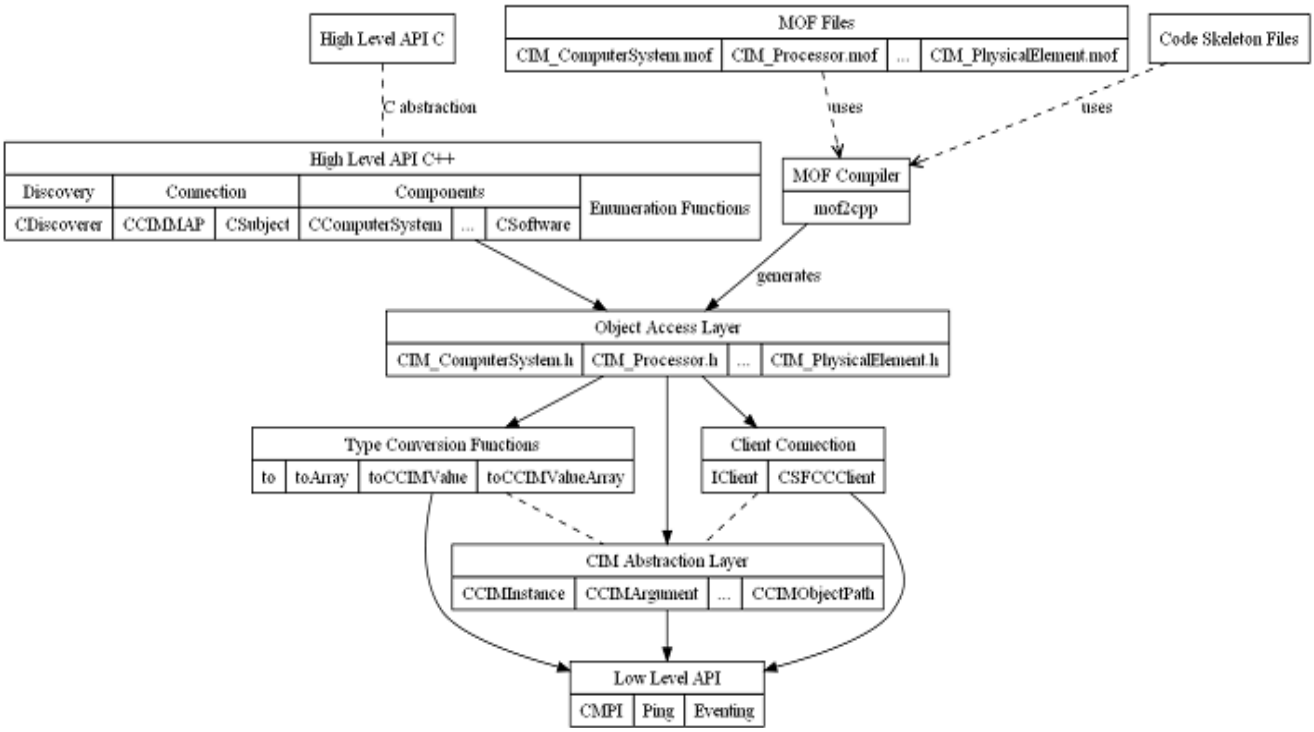


* The DASH Client SDK implements a WSMan interface to the DASH Server, as required by the DASH Implementation Requirements Specification. Other interfaces, such as CIM/XML, and other WS-Man stacks, can be bound to the Low Level API. The details of how to bind the Low Level API to another interface are beyond the scope of this document.

2 Design and Features

Design details

2.1 High Level SDK design



SDK Structure

The SDK is layered to provide clean interfaces for consumers of DASH functions and to make the SDK easy to extend. Users of the SDK will use the High Level API with C and C++ bindings, which are introduced below and described in detail in Sections 3 and 4 of this document. Developers who wish to add new profiles and new use cases to the SDK will modify the Components layer, perhaps adding new Enumeration Functions, and likely adding new MOF files to represent additional CIM classes. Developers who want to create new transport bindings for the SDK (to use a different WS-Man stack, for example) will modify the bindings at the Low Level API layer.

Low-Level API

The Low-Level API is described in Section 5 of this document.

CIM Abstraction Layer

The Common Information Model Abstraction layer provides an Object Oriented Abstraction over the CMPI objects. The objects exposed by this layer perform the resource cleanup on their deletion keeping the upper layers easier to implement.

The resources contained within this objects are reference counted and hence these objects can be passed by value without much overhead. This helps implementing the higher layers easier and prevents unnecessary memory leaks.

Type Conversion Functions

Values of CIM methods, parameters, properties and qualifiers are abstracted in the C++ class CCIMValue. Type Conversion functions convert from CCIMValue to the native types and vice versa. These functions perform the type checking and throw an exception if the type conversion is not valid. NOTE: WS-Man does not carry value type information. The Type Conversion functions validate only that value strings carried in a WS-Man message can be converted to CIM types.

Client Connection

This layer provides an abstract interface to access the client.

Object Access Layer

This layer provides a C++ representation of CIM objects. The classes are generated from the corresponding mof files using the mof2cpp compiler, creating C++ classes that correspond to CIM classes. The properties of the objects in this layer are cacheable i.e. you can mark the properties that you would like to access as cached and they will be fetched together transparently when any one property is fetched. After this any access to a property is always got from the cached value until the cache is invalidate, a property set or method invoked.

The properties setter/getter, methods of the objects in this layer accept only native types that match the types specified in the MOF files. This makes any higher level access type safe. Also the objects in this layer provides the enums for any value types defined in the MOF file that can be used safely by the higher layers instead of using a hardcoded value. The higher layers can also obtain the string values corresponding to the enum values by making appropriate function calls.

This is the best layer for any software to use if it requires more features than the DASH functionalities offered by the high level API.

MOF to CPP Code Generator

This compiler parses the MOF file provided to it and converts to C++ code. A code skeleton file directory is provided as input to this compiler to generate the code.

Features

The DASH SDK main features include Low-Level API, High-Level API, and CLI.These features are introduced below and described in more detail later in this document.

2.2 Low-Level API

The Low-level API is based on the Common Manageability Programming Interface (**CMPI**) with additional functions to perform DASH ping; event subscription; and event listening and reporting. An important characteristic of the Low-Level API is that it provides a direct mapping to the Common Information Model-CIM operations (sometimes described as "generic operations") that are commonly implemented over CIM-XML protocols and which are being standardized by DMTF. These operations include

- Read and write access to CIM instances and their properties and methods.
- Getting and setting more than one property of an instance atomically.
- Enumeration and query of CIM instances.

The Low-level API also has a function that enables upper layer software to send a raw WS-Man XML command (i.e. a SOAP document with WS-Management headers). This capability may be useful for debugging and for binding the SDK to proprietary interfaces.

CMPI is a standard interface defined by The Open Group (http://www.opengroup.org). The DASH SDK extensions, which map CMPI operations to WSMan actions, are described in Section 5 of this document.

2.3 High-Level API

The high level API is defined for common use cases on DASH targets such a discovery, power on/off, basic inventory, etc. The high level API is an abstraction layer that minimizes the exposure of SDK users to the details of the DASH protocol stack. The High level API is provided both as C and C++ interfaces. The main characteristics of the High-Level API include:

- The High-level API provides APIs for the most common user-level operations or use-cases that are supported by DASH 1.0. These high-level APIs provide additional abstraction of the details of the CIM and profiles. These APIs completely abstract details of the CIM and profiles from the caller.
- High-Level APIs include:
    - DASH 2-phase discovery—executes the 2-phase discovery and returns protocols (including DASH) that are supported and capabilities
    - Power control (power-status/on/off)
    - Get basic inventory—including memory, processor information, platform vendor and model, OS, hostname, etc.
    - User management (get user list, create user, configure user)
    - Boot path selection setting(changing boot order etc)
    - Common inventory properties ~~e.g.~~ (Computer name, firmware revision, etc.)

2.4 High-Level API (in C)

This is an abstraction of the C++ high level API in C language. The objects in this interface consist of an opaque pointer (this corresponds to the C++ objects) and a corresponding function table (this corresponds to the objects access functions). The functions in the function table delegate the calls to the appropriate C++ functions.

2.5 CLI

The Command Level Interface provides a scripting interface to the C++ High Level DASH API. The CLI is provided with sub commands and targets. Each target has its own sub-commands that are specific to that particular target. Main characteristics of CLI includes

- The DASH SDK CLI provides full CLI access to the public Low-and High-Level APIs. This provides:
    - A vehicle to test the Low-Level APIs
    - Debug/development tools to test real implementations
    - Reference code for how to use the public interfaces.

- The DASH SDK CLI provides CLI access to all of the abstracted features in the High-Level API.
- It can be run as a shell.
- CLI supported commands are shown in the help screen shown below:


Command Prompt
C:\Program Files\Dash Software Development Kit\source\release>dashcli.exe help
Usage: dashcli [options] commands
Options allowed:
-h <host> Host Name
-p <port(s)> Server Port(s)(For discovery more than one ports can be specified seperated by commas)
-u <username> User Name
-P <password> Password
-a <digest|basic|gss> Authentication Type [default=digest]
-S <http|https> HTTP Scheme [default=http]
-C Ignore certificate/do not verify certificate (To verify, certificate should be stored in cert
-t <targetpath> Target Path
-s <startip> Start IP address for discovery (only for discovery)
-e <endip> End IP address for discovery (only for discovery)
-T <timeout> Timeout in seconds
-v <1|2> Verbose Level [1 - More explanation on error or 2-Dump WSMAN data]
-o <verboseoutput> Verbose output file to dump wsman data [default is sdtout].
Commands allowed:
help Display help
enumerate Enumerate targets
show Show dashcli information
discover Perform discovery
registeredprofile Checks the profile support
indication Indication commands(subscribe for indication, create filters/destinations
listenevents Listen for events/alerts
textredirection Configure Text Redirection services
usbredirection Configure USB Redirection services
raw Issue raw commands
account Creates,Deletes and Manages the Account
roles Creates,Deletes and Manages the Roles
shell Launch interactive DASH shell
capabilities Display Capabilities of a target
For commands specific to targets
dashcli help target
Where allowed targets are
registeredprofile
computersystem
processor
memory
asset
bootconfig
bios
powersupply
fan
software
operatingsystem
battery
role
networkport
dhcpclient
ipinterface
dnsclient
opaquemanagementdata
indicationsubscription
ethernetport
Example usage:
Discovery example:
dashcli -s 192.168.0.4 -e 192.168.0.15 -u admin -P admin -p 623 discover
dashcli -s 192.168.0.4 -e 192.168.0.15 -p 623 discover
dashcli -s 192.168.0.4 -e 192.168.0.15 -p 623,664,8889 discover
dashcli -s 192.168.0.4 -e 192.168.0.15 -S http -p 623 discover
dashcli -s 192.168.0.4 -e 192.168.0.15 -S https -p 664 discover
dashcli -s 192.168.0.4 -e 192.168.0.15 discover info

3 High Level API-C++

The high level Application Programming Interface is defined for common operations on a DASH target such as discovery, power on/off, basic inventory, etc. The high level API is an abstraction layer that minimizes the exposure of the console to the details of the Low-Level DASH protocol stack. It is provided in both a C and C++ interfaces.
C++ APIs are categorized into three main groups. These are listed below:

3.3.29 CIndication Subscription

These classes are discussed in more details in the following sections. All the C++ classes are scoped in the namespace dsdk.

3.1. Discovery Classes

3.1.1. Cdiscoverer

The CDiscoverer class allows the upper layer to discover the CIM Management Access Points present in a network.

Member Functions

This class has two member functions that are listed below.

- discoverMAP
- discoverMAPs

Member Functions Description

The following section provides more details on the class member functions

- **discoverMAPs**

  This functions discovers the management access point in the network range between a given start_ip and end_ip addresses

  Syntax:
  ```
  vector<CCIMMAP> discoverMAPs(const string& start_ip,
                               const string& end_ip,
                               const vector <pair <string,
                               string> >&port_httpscheme ,
                               u32 timeout);
  ```

  Parameters:

  - *start_ip:* Starting IP Address from which to search from. This must be an IPV4 address, expressed as dot-separated integers. Example: "10.10.37.1".
  - *end_ip:* Ending IP Address at which the search should end. If this value is empty then only the start_ip is searched. If this address is a numerically lower address than the s*tart_ip* value, then this method will probably never return, because it will search all the addresses from start_ip to 255.255.255.254 and then all from 0.0.0.1 to end_ip. *port_httpshceme* Http Protocol..If not specified, http protocol will be used by default.If it is blank(""), default protocol for the ports will be used (https for 664 and http for other ports including 663). If this vector is filled in, the pair of strings is "port" and "scheme". Example: pair<"16992", "https">.

- *Timeout :* Timeout in seconds. The default value of 5 seconds is usually sufficient, but note that the API will try each *port_httpscheme* pair serially. If there are two port/schemes in the list, and no DASH MAP at either, then the total default timeout will be 10 seconds. If a value is specified, it must be greater than 0.

This function returns the list of the discovered CIMMAP. When no management access point are found, returned CIMMAP vector size will 0.

- **discoverMAP.**

  Description: Discover the management access point that is present in the host_name.

  Syntax:
  ```
  vector<CCIMMAP> discoverMAP (const string& hostname,
                               const string& port,
                               const string& http_scheme,
                               u32 timeout);
  ```

  Parameters:
  - *host_name :* The host name to get the CIMMAP. This may be a resolvable host name or an IPV4 address. Examples: "mypc", "10.55.123.7").
  - *port :* Port number. Note : specify the default behavior.
  - *http_scheme b :* Http Protocol, Note : specify the default behavior.
  - *Timeout :* Timeout in seconds. Note: specify the default and the behavior when a value of 0 is specified.

  This function returns The list CIMMAP that is being discovered (This is a list because there could be two MAPs at 623 & 664).

  Note:
  All Member functions in CDiscoverer API throw exceptions if error occurs.
  Following exceptions are thrown by this API.
  - EDSDKError.
  - ECIMError

  The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- discoverer.h
Library -- dashapi

Usage Examples

Below is an example on how this function can be used by an application. This example discovers the the management access point from the IP address 192.168.0.10 to 192.168.0.20 on

ports 623, 664. If not specified, http protocol will be used by default.If it is blank(“”),default protocol for the ports will be used (https for 664 and http for other ports including 663).

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "discoverer.h"

using namespace dsdk;

int main (void)
{

        /* add the list of ports to discover */

        vector < pair <string, string> v_port_scheme;
        pair <string, string> port_scheme;
        port_scheme.first = "623";
        port_scheme.second = "http";
        v_port_scheme.pushback (port_scheme);

        port_scheme.first = "664";
        port_scheme.second = "https";
        v_port_scheme.pushback (port_scheme);

        port_scheme.first = "664";
        port_scheme.second = "";
        v_port_scheme.pushback (port_scheme);

        port_scheme.first = "623";
        port_scheme.second = "";
        v_port_scheme.pushback (port_scheme);

        try
        {
                /*do the discovery */
                vector<CCIMMAP> cimmap = CDiscoverer::discoverMAPs ("192.168.0.10",
                                                                    "192.168.0.20",&v_port_s
                                                                chemes, timeout);

                /* display the discovered management access points */
                fprintf (stdout, "Discovered:\n");
                for (size_t i = 0; i < cimmap.size (); i++)
                {
                        fprintf (stdout, "\t%s:%s\n", cimmap [i].getHostName ().c_str (),
                                                        cimmap [i].getPort ().c_str ());
                }
        }
        catch (exception& e)
```

```
        {
                fprintf (stdout, "Error : %s\n", e.what());
        }

        return 0;
}
```

Below is a sample output for above example

```
Discovered :
                192.168.0.11:623
Discovered :
                192.168.0.11:664
Discovered :
                192.168.0.11:664
Discovered :
                192.168.0.11:623
```

Usage Notes: In most cases, searching a large number of IP addresses for DASH MAPs is inefficient and non-scalable. The CDiscoverer APIs are provided for situations when these problems are tractable.

One way to slightly improve the scalability of discovery is to break the search space into smaller ranges and call the DiscoverMAPS method for the smaller ranges in parallel threads.

"**Please note that a broadcast address within the range will be treated as any other address during the discovery.**"

3.2. Connection Classes

3.2.1. CCIMMAP

This is a class that abstracts a DASH Management Access Point (MAP).

Member Functions

Functions member of this class consist of the following:

- CCIMMAP: This is constructor.
- connect
- getHostName
- getPort

Constructor Description

- **CCIMMAP**
  The CCIMMAP Constructs the CIMMAP object.

Syntax: CCIMMAP (string host_name,string port = "8888");

Parameters:

- *host_name* Host name
- *port* Port Number, when not specified uses the default port(8888).

Member Functions Description

- **connect**

  This function creates a logical connection to a DASH Server using the credentials supplied in subject. The logical connection is encapsulated in an object that implements the IClient interface.

  Syntax: IClient* connect(const CSubject & subject);

  Parameters:

  - *subject* Credentials to connect to the DASH server.

  This function returns a client handle (pointer)

  Notes:

  1. **The returned IClient object must be destroyed using the delete operator when it is no longer needed.**
  2. **The current implementation does not check that the logical connection represented by the IClient is usable. In fact, no network traffic is generated when the connect() method is invoked. Attempts to interact with a MAP using this object may fail for any number of reasons, the most likely of which is that the combination of IP address, port, http scheme, user name and password is incorrect. To determine whether an IClient object returned from the connect() method is valid, try using it in an operation as shown in the sample code.**

- **getHostName**

  Description: Gets the host name

  syntax: string getHostName (void);

  Returns: The host name

- **getPort**

  Description: Gets the port number.

  Syntax: string getPort (void);

  Returns: The port number.

  Note:

  All Member functions in CCIMMAP throw exceptions if error occurs.

  Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EConnectionFailed

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirement

Header file -- cimmap.h
Library -- dashapi

Usage Examples

Below are two examples on how to use this API.

Example 1: This example connects to a MAP specified by the IP address and display the computer systems name.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
main (void)
{

        CCIMMAP  cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin", "digest");
        IClient* client = cm.connect (subject);


        try
        {
                /* ------- do your stuff ------ */
                CComputerSystem::iterator iter
                                        = CComputerSystem::enumComputerSystems (client);
                for (; iter != CComputerSystem::iterator::end (); ++iter)
                {
                        /* get the instances of the computer system */
                        CComputerSystem cs = *iter;
                        fprintf (stdout, "Computer System Name : %s\n", cs.getName
                                                                        ().c_str ());
                }
```

```
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error : %s\n", e.what());
        }


        delete client;

        return 0;
}
```

Below is sample output for the above example

```
Computer System Name : mkl-desktop
```

Example 2: This example connects discovered targets and enumerate the computer system with in the target..

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "discoverer.h"
#include "computersystem.h"

using namespace dsdk;

int main (void)
{
        vector <string> ports;

        /* add the list of ports to discover */
        ports.push_back ("623");

        try
        {
                /*do the discovery */
                vector<CCIMMAP> cimmap = CDiscoverer::discoverMAPs ("192.168.0.10",
                                                                    "192.168.0.20",
                                                                    ports);
                for (size_t i = 0; i < cimmap.size (); i++)
                {
                        CSubject subject ("admin", "admin","digest");
                        IClient* client = cimmap [i].connect (subject);
```

```
                        /* ------- do your stuff ------ */
                        CComputerSystem::iterator iter =
                                        CComputerSystem::enumComputerSystems (client);

                        delete client;
                }
        }
        catch (exception& e)
        {
                fprintf (stdout, "Error : %s\n", e.what());
        }

        return 0;
}
```

3.2.2. CSubject

A class representing credentials/subject.

Constructor

- **CSubject**

  Description: Constructs a Subject object.

  Syntax: CSubject (string user, char* password, string auth, unsigned long timeout = 0)

  Parameters:
  - *user* User name
  - *password* Password
  - *auth* Authentication type. Supported types are *basic* and *digest*
  - *timeout* Time out in seconds, if not specified uses the default time out value.

  Note:
  All Member functions in CSubject throw exceptions if error occurs.
  Following exceptions are thrown by this API.
  - EDSDKError.
  - ECIMError
  - ENotEnoughMemory

  The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- subject.h

Library -- dashapi

Usage Examples

Below example on how to use this API.

Example 1: This example connects to a MAP specified by the IP address and display the computer systems name.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
main (void)
{

        CCIMMAP  cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin", "digest");
        IClient* client = cm.connect (subject);


        try
        {
                /* ------- do your stuff ------ */
                CComputerSystem::iterator iter
                                        = CComputerSystem::enumComputerSystems (client);
                for (; iter != CComputerSystem::iterator::end (); ++iter)
                {
                        /* get the instances of the computer system */
                        CComputerSystem cs = *iter;
                        fprintf (stdout, "Computer System Name : %s\n", cs.getName
                                                                        ().c_str ());
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error : %s\n", e.what());
        }

        delete client;

        return 0;
}
```

Below is sample output for the above example when tested against RI.

```
Computer System Name : mkl-desktop
```

3.3 Component Classes

3.3.1. CComputerSystem

A class that represents a computer system.

Member Functions

The class member functions consist of of the following:

- getCIMObject
- getName
- getPrimaryOwner
- getPrimaryOwnerContact
- getReqPwrStateChangeErrStr
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getDedicated
- getDedicatedStr
- getPowerState
- getPowerStateStr
- getRequestedPowerState
- getRequestedPowerStateStr
- getPowerOnTime
- getPowerChangeCapabilities
- getPowerChangeCapabilitiesStr
- getPowerStatesSupported
- getPowerStatesSupportedStr
- getReqPwrStateChangeErrStr
- capableOfPowerStatesManagement
- capableOfRequestPowerStateChange
- isSupportedValue
- powerOn

- powerOff
- powerCycle
- powerReset

Static Member Functions

- enumComputerSystems
- getCachedProps

Constructors Description

- **CComputerSystem**

  Description: Constructs this object from the corresponding CIM_ComputerSystem object.

  Syntax: CComputerSystem (const CIM_ComputerSystem & cs);

  Parameters:
  - *cs* CIM Computer System object.

Member Functions Description

- **enumComputerSystems**

  Description: Enumerates all computer systems present under a management access point.

  Syntax: CComputerSystem::iterator enumComputerSystems (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *Cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the computer systems.

- getCachedProps

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_ComputerSystem* getCIMObject (void);

  Returns: Returns the underlying CIM object.

- **getName**

  Description: Gets the name of this computer system

  Syntax: string getName (void);

  Returns: The name of the computer system.

- **getPrimaryOwner**
  Description: Gets the primary owner of the computer system
  Syntax: string getPrimaryOwner (void);
  Returns: The primary owner of the computer system.

- **getPrimaryOwnerContact**
  Description: Gets the primary owner contact of the computer system
  Syntax: string getPrimaryOwnerContact (void);
  Returns: The primary owner contact of the computer system.

- **getReqPwrStateChangeErrStr**
  Description: Gets the error description for request change power state error code.
  Syntax: string getReqPwrStateChangeErrStr (uint32 err);
  Parameters:
    - *err* Error code returned from change power state command(Power on,power reset)
  Returns: The error description.

- **getEnabledState**
  Description: Gets the EnabledState of the computer system
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState of the computer system as string.
  Syntax: string getEnabledStateStr (void) const;
  Return: The EnabledState.

- **getRequestedState**
  Description: Gets the last RequestedState of the computer system.
  Syntax: uint16 getRequestedState (void) const;
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the last RequestedState of the computer system as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The RequestedState

- **getDedicated**
  Description: Gets the purpose(s) of this computer is dedicated to.
  Syntax: vector<uint16> getDedicated (void) const;

Returns: The dedicated purpose(s)

- **getDedicatedStr**
  Description: Gets the purpose(s) of this computer is dedicated to as string.
  Syntax: vector<string> getDedicatedStr (void) const;
  Returns: The dedicated purpose(s)

- **getPowerState**
  Description: Gets the current power state of this computer system.
  Syntax: uint16 getPowerState (void) const;
  Returns: The current power state.

- **getPowerStateStr**
  Description: Gets the power state of this computer system as a string.
  Syntax: string getPowerStateStr (void) const;
  Returns: The power state as string

- **getRequestedPowerState**
  Description: Gets the last requested power state of this computer system.
  Syntax: uint16 getRequestedPowerState (void) const;
  Returns: The last requested power state.

- **getRequestedPowerStateStr**
  Description: Gets the power state as a string.
  Syntax: string getRequestedPowerStateStr (void) const;
  Returns: The power state as string

- **getPowerOnTime**
  Description: Gets the time when this computer system will be powered on again.
  Syntax: datetime getPowerOnTime (void) const;
  Returns: The next power on time.

- **getPowerChangeCapabilities**
  Description: Gets the power change capabilities of this computer system.
  Syntax: vector<uint16> getPowerChangeCapabilities (void) const;
  Returns: The power change capabilities

- **getPowerChangeCapabilitiesStr**
  Description: Gets the power change capabilities of this computer system as string
  Syntax: vector<string> getPowerChangeCapabilitiesStr (void) const;
  Returns: The power change capabilities

- **getPowerStatesSupported**
  Description: Gets the power states supported for this computer system.
  Syntax: vector<uint16> getPowerStatesSupported (void) const;
  Returns: The power state supported

- **getPowerStatesSupportedStr**
  Description: Gets the power states supported for this computer system as string.
  Syntax: vector<string> getPowerStatesSupportedStr (void) const;
  Returns: The power state supported

- **getReqPwrStateChangeErrStr**
  Description: getReqPwrStateChangeErrStr
  Syntax: string getReqPwrStateChangeErrStr (uint32 err) const;
  Parameters:
    - *err* *error*
  Returns: The power state change error as string

- **capableOfPowerStatesManagement**
  Description: Verifies whether the PowerManagementService and PowerManagementCapbilities instance exists or not.
- Syntax: ***bool capableOfPowerStatesManagement (void) const;***
  Returns: True or False

- **capableOfRequestPowerStateChange**
  Description: Verifies whether the PowerStateChange operation is supported or not.
  Syntax: ***bool capableOfReuestPowerStateChange (void) const;***
  Returns: True or False

- **isSupportedValue**
  Description: Verifies whether the particular Power State property value is supported or not. (E.g. On, Off, Reset, PowerCycle etc.)
  Syntax: ***bool isSupportedValue (uint16 val) const;***
  Returns: True or False

- **powerOn**
  Description: Power On the computer system
  Syntax: uint32 powerOn (void);
  Returns: 0 if success, error code if failed. Use getReqPwrStateChangeErrStr to get the error description.

- **powerOff**
  Description: Power off the computer system.
  Syntax: uint32 powerOff (void) const;
  Returns: 0 if success, error code if failed. Use getReqPwrStateChangeErrStr to get the error description.

- **powerCycle**
  Description: PowerCycle the computer system
  Syntax: uint32 powerCycle (void) const;
  Returns: 0 if success, error code if failed. Use getReqPwrStateChangeErrStr to get the error description.

- **powerReset**
  Description: PowerReset the computer system
  Syntax: uint32 powerReset (void) const;
  Returns: 0 if success, error code if failed. Use getReqPwrStateChangeErrStr to get the error description.

Note: All the Member function in CComputerSystem throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionNotSupported
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- computersystem.h
Library -- dashapi

Usage Examples

Enumerate the computer systems and display the name and power on the computer system.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
```

```
main (void)
{
	CCIMMAP cm ("192.168.0.20", "623");
	CSubject subject ("admin", "admin","digest");
	IClient* client = cm.connect (subject);

	try
	{
		/* enumerate the computer systems */
		CComputerSystem::iterator i = CComputerSystem::enumComputerSystems
									(client);
		for (; i != CComputerSystem::iterator::end (); ++i)
		{
			/* get the instances of the computer system */
			CComputerSystem cs = *i;

			/* get the name of computer system */
			try
			{
				fprintf (stdout, "Computer system name : %s\n", cs.getName
									().c_str ());
			}
			catch (EDSDKError &e)
			{
				fprintf (stdout, "Error getting computer system name :
									%s\n", e.what());
			}
			catch (exception &e)
			{
				fprintf (stdout, "Error getting computer system name :
									%s\n", e.what());
			}

			try
			{
				/* power on the computer system */
				cs.powerOn ();
				fprintf (stdout, "Power on success\n");
			}
			catch (EDSDKError &e)
			{
				fprintf (stdout, "Error power on : %s\n", e.what());
			}
			catch (exception &e)
			{
				fprintf (stdout, "Error power on : %s\n", e.what());
			}
		}
	}
	catch (exception &e)
```

```
        {
                fprintf (stdout, "Error accessing computer system : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

Below is a sample out put for the above example when run against RI.

```
Computer system name : mkl-desktop
Power on success
```

3.3.2. CFan

This is a class that represents a fan.

Member Functions

- getCIMObject
- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- isVariableSpeed
- getSpeed
- getDesiredSpeed
- setDesiredSpeed
- isActiveCooling
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getElementName
- getSetSpeedErrStr
- capableOfSetFanSpeed
- capableOfRequestStateChange

- getSupportedStates
- getSupportedStatesStr

Static Member Functions

- enumFans
- getCachedProps

Constructors Description

- **CFan**

  Description: Construct this object from the corresponding CIM_Fan object

  Syntax: CFan (const CIM_Fan& fan);

  Parameters:

- *fan* CIM fan object;

Member Functions Description

- **enumFans**

  Description: Enumerates all fans present under a Management Access Point.

  Syntax: CFan::iterator enumFans (IClient* client, bool cached = true);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the fan.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_Fan* getCIMObject (void);

  Returns: Returns the underlying CIM object.

- **getSystemCreationClassName**

  Description: Gets the System Creation class name of the fan

  Syntax: string getSystemCreationClassName (void) const;

  Returns: The System Creation Class name

- **getSystemName**

  Description: Gets the System name of the fan

  Syntax: string getSystemName (void) const;

Return: The System name

- **getCreationClassName**
  Description: Gets the Creation class Name of the fan
  Syntax: string getCreationClassName (void) const;
  Returns: The fan creation class name.

- **getDeviceID**
  Description: Gets the device id of the fan
  Syntax: string getDeviceID (void) const;
  Returns: The fan device id.

- **getOperationalStatus**
  Description: Gets the operational status of the fan
  Syntax: vector<uint16> getOperationalStatus (void) const;
  Returns: The operational status

- **getOperationalStatusStr**
  Description: Gets the operational status of the fan as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: The operational status

- **getHealthState**
  Description: Gets the health state of the fan
  Syntax: uint16 getHealthState (void) const;
  Returns: The health state

- **getHealthStateStr**
  Description: Gets the health state of the fan as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The health state

- **isVariableSpeed**
  Description: Returns true if the fan supports variable speed
  Syntax: bool isVariableSpeed (void) const;
  Returns: true if fan supports variable speed . false otherwise.

- **getSpeed**
  Description: Gets the fan's current speed(tach reading).It returns the Reading in RPM . If tach sensor is analog(numeric) sensor, else returns the status if its discrete sensor.
  Syntax: string getSpeed(void) const;

Returns: The speed of the fan.

- **getDesiredSpeed**
  Description: Gets the desired speed of the fan
  Syntax: uint64 getDesiredSpeed (void) const;
  Returns: The desired speed of the fan.

- **setDesiredSpeed**
  Description: Sets the desired speed of the fan
  Syntax: uint32 setDesiredSpeed (uint64 speed) const;
  Parameters:
  - *speed* The desired speed.

  Returns: 0 if success, error code if failure.

- **isActiveCooling**
  Description: Checks if the fan supports active cooling speed.
  Syntax: uint16 getEnabledState (void) const;
  Returns: True if its active cooling, else false.

- **getEnabledState**
  Description: Gets the state of the Fan
  Syntax: uint16 getEnabledState (void) const;
  Returns: The enabled state

- **getEnabledStateStr**
  Description: Gets the state of the Fan as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The enabled state

- **getRequestedState**
  Description: Gets the last requested state of the Fan
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requested state

- **getRequestedStateStr**
  Description: Gets the last requested state of the Fan as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The requested state

- **capableOfSetFanSpeed**
  Description: Verifies whether SetFanSpeed operation can be performed or not.

Syntax: ***bool capableOfSetFanSpeed (void) const;***
Returns: True or False

- **capableOfRequestStateChange**
  Description: Verifies whether the Fan State Management is supported or not.
  Syntax: ***bool capableOfRequestStateChange (void) const;***
  Returns: The requested state

- **getSupportedStates**
  Description: Gets the Supported States of the Fan as integer value
  Syntax: ***vector<uint16> getRequestedStateStr (void) const;***
  Returns: The supported states

- **getSupportedStatesStr**
  Description: Gets the Supported States of the Fan as String value
  Syntax: ***vector<string> getRequestedStateStr (void) const;***
  Returns: The supported states

- **getElementName**
  Description: Returns the name of the Element
  Syntax: ***string getElementName(void) const;***
  Returns: The ElementName.

- **getSetSpeedErrStr**
  Description: Get Set SpeedErrStr
  Syntax: string getSetSpeedErrStr (uint32 err) const;
  Parameters:
  - *err* Error code returned from setDesiredSpeed

  Returns: Error description.

Note: All the Member function in CFan throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirement

Header file -- fan.h
Library -- dashapi

Usage Examples

Enumerate the fan and display the speed.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "fan.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        /* enumerate the fans */
        CFan::iterator i = CFan::enumFans (client);
        for (; i != CFan::iterator::end (); ++i)
        {
                /* get the instances of the fan */
                CFan fan = *i;

                try
                {
                        /* get the fan speed */
                        fan.getDesiredSpeed ();
                }
                catch (...)
                {
                        fprintf (stdout, "Error getting fan speed \n");
                }
        }

        delete client;

        return 0;
}
```

3.3.3. CPhysicalAsset

This class represent a Physical Asset.

Member Functions

- getCIMObject
- getTag
- getCreationClassName
- getManufacturer
- getModel
- getSerialNumber
- getPartNumber
- getSKU
- getElementName
- canBeFRUed
- getPackageType
- getPackageTypeStr
- getVendorCompatibilityStrings
- getVersion
- getName
- isHostingBoard
- getTypeOfRack
- getTypeOfRackStr
- getChassisPackageType
- getChassisPackageTypeStr
- getConnectorLayout
- getConnectorLayoutStr
- getSlotNumber
- getFormFactor
- getMemoryType
- getMemoryTypeStr
- getMemorySpeed
- getMemoryCapacity
- getMemoryBankLabel

Static Member Functions

- enumPhysicalAssets
- getCachedProps

Constructors Description

- **CPhysicalAsset**

  Description: Construct this object from the corresponding CIM_PhysicalElement object.

  Syntax: CPhysicalAsset (const CIM_PhysicalElement & pe);

Parameters:

- *pe* CIM Physical Element object.

Member Functions Description

- **enumPhysicalAssets**

  Description: Enumerates all the physical assets that are present under a Management Access Point.

  Syntax: CPhysicalAsset::iterator enumPhysicalAssets (IClient* client, bool cached = true);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the physical asset.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_PhysicalElement* getCIMObject (void);

  Returns: The underlying CIM object.

- **getTag**

  Description: Gets the physical asset Tag

  Syntax: string getTag (void) const;

  Returns: The physical asset Tag

- **getCreationClassName**

  Description: Gets the physical asset CreationClassName

  Syntax: string getCreationClassName (void) const;

  Returns: The physical asset CreationClassName

- **getManufacturer**

  Description: Get the manufacturer of the physical asset.

  Syntax: string getManufacturer (void) const;

  Returns: The manufacturer string.

- **getModel**

  Description: Gets the model of the physical asset.

  Syntax: string getModel (void) const;

Returns: The model string.

- **getSerialNumber**
  Description: Gets the serial number of the physical asset.
  Syntax: string getSerialNumber (void) const;
  Returns: The serial number string.

- **getPartNumber**
  Description: Gets the part number of the physical asset.
  Syntax: string getPartNumber (void) const;
  Returns: The part number string.

- **getSKU**
  Description: Gets the SKU info of the physical asset.
  Syntax: string getSKU (void) const;
  Returns: The SKU String.

- **getElementName**
  Description: Gets the ElementName.
  Syntax: string getElementName (void) const;
  Returns: The ElementName.

- **canBeFRUed**
  Description: Check if physical asset can be FRUed.
  Syntax: boolean canBeFRUed (void) const;
  Returns: If physical asset can be FRUed

- **getPackageType**
  Description: Gets the package type
  Syntax: uint16 getPackageType (void) const;
  Returns: The package type.

- **getPackageTypeStr**
  Description: Gets the package type as string
  Syntax: string getPackageTypeStr (void) const;
  Returns: The package type.

- **getVendorCompatibilityStrings**
  Description: Gets the VendorCompatibilityStrings
  Syntax: vector<string> getVendorCompatibilityStrings (void) const;
  Returns: The VendorCompatibilityStrings

- **getVersion**
  Description: Gets the version of physical package
  Syntax: string getVersion (void) const;
  Returns: The version

- **getName**
  Description: Get the name of physical package
  Syntax: string getName (void) const;
  Returns: The name

- **isHostingBoard**
  Description: Gets the card's HostingBoard
  Syntax: boolean isHostingBoard (void) const;
  Returns: The HostingBoard

- **getTypeOfRack**
  Description: Gets the Rack type of the Rack as string
  Syntax: string getTypeOfRackStr (void) const;
  Returns: The TypeOfRack

- **getChassisPackageType**
  Description: Gets the chassis package type
  Syntax: uint16 getChassisPackageType (void) const;
  Returns: The ChassisPackageType

- **getChassisPackageTypeStr**
  Description: Gets the chassis package type as string
  Syntax: string getChassisPackageTypeStr (void) const;
  Returns: The ChassisPackageType

- **getConnectorLayout**
  Description: Gets the physical connector layout
  Syntax: uint16 getConnectorLayout (void) const;
  Returns: The ConnectorLayout

- **getConnectorLayoutStr**
  Description: Gets the physical connector layout as string
  Syntax: string getConnectorLayoutStr (void) const;
  Returns: The ConnectorLayout

- **getSlotNumber**
  Description: Gets the slot number

Syntax: uint16 getSlotNumber (void) const;
Returns: The slot number

- **getFormFactor**
  Description: Gets the memory form factor
  Syntax: uint16 getFormFactor (void) const;
  Returns: The FormFactor

- **getMemoryType**
  Description: Gets the memory type
  Syntax: uint16 getMemoryType (void) const;
  Returns: The MemoryType

- **getMemoryTypeStr**
  Description: Gets the memory type as string
  Syntax: string getMemoryTypeStr (void) const;
  Returns: The MemoryType

- **getMemorySpeed**
  Description: Gets the memory speed
  Syntax: uint32 getMemorySpeed (void) const;
  Returns: The memory speed

- **getMemoryCapacity**
  Description: Gets the memory capacity
  Syntax: uint64 getMemoryCapacity (void) const;
  Returns: The memory capacity

- **getMemoryBankLabel**
  Description: Gets the memory Bank Label
  Syntax: string getMemoryBankLabel (void) const;
  Returns: The memory BankLabel

Note: All the Member function in CPhysicalAsset throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- physicalasset.h
Library -- dashapi

Usage Examples

Enumerate the physical asset and display the asset information

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "physicalasset.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        /* enumerate the assets */
        CPhysicalAsset::iterator i = CPhysicalAsset::enumPhysicalAssets (client);
        for (; i != CPhysicalAsset::iterator::end (); ++i)
        {
                /* get the instances of the asset*/
                CPhysicalAsset pa = *i;

                try
                {
                        /* get the asset information */
                        pa.getManufacturer ();
                        pa.getVersion ();
                        pa.getSerialNumber ();
                        pa.getModel ();
                }
                catch (...)
                {
                        fprintf (stdout, "Error accessing physical asset
                                                                        informantion\n");
                }
        }

        delete client;

        return 0;
}
```

3.3.4. CPhysicalMemory

This is a class that represents a PhysicalMemory.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- isVolatile
- getAccess
- getAccessStr
- getBlockSize
- getNumberOfBlocks
- getConsumableBlocks
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getElementName

Static Member Functions

- enumPhysicalMemory
- getCachedProps

Constructors Description

- **CPhysicalMemory**
  Description: Constructs this object from the corresponding CIM_PhysicalMemory object.
  Syntax: CPhysicalMemory (const CIM_PhysicalMemory & pm);
  Parameters:
  - *pm* CIM Physical Memory object.

Member Functions Description

- **enumPhysicalMemory**

Description: Enumerates all the physical memory present under a Management Access Point.

Syntax: CPhysicalMemory::iterator enumPhysicalMemory (IClient* client, bool cached = true);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching. Default is true.

Returns: Iterator to the physical memory.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_PhysicalMemory* getCIMObject (void);

  Returns: The underlying CIM object

- **getSystemCreationClassName**

  Description: Gets the memory SystemCreationClassName

  Syntax: string getSystemCreationClassName (void) const;

  Returns: The memory SystemCreationClassName

- **getSystemName**

  Description: Gets the memory CreationClassName

  Syntax: string getCreationClassName (void) con**st;**

  Returns: The memory CreationClassName

- **getDeviceID**

  Description: Gets the memory DeviceID

  Syntax: string getDeviceID (void) const;

  Returns: The memory DeviceID.

- **isVolatile**

  Description: Checks if memory is Volatile

  Syntax: boolean isVolatile (void) const;

  Returns: true

- **getAccess**

  Description: Gets the memory access type Ex (Read/Write etc).

  Syntax: uint16 getAccess (void) const;

Returns: The memory access.

- **getAccessStr**
  Description: Gets the memory access type as string Ex (Read/Write etc).
  Syntax: string getAccessStr (void) const;
  Returns: The memory access.
- **getBlockSize**
  Description: Gets the BlockSize of memory
  Syntax: uint64 getBlockSize (void) const;
  Returns: The memory BlockSize.

- **getNumberOfBlocks**
  Description: Gets the NumberOfBlocks of memory
  Syntax: uint64 getNumberOfBlocks (void) const;
  Returns: The NumberOfBlocks.

- **getConsumableBlocks**
  Description: Gets the ConsumableBlocks of memory
  Syntax: uint64 getConsumableBlocks (void) const;
  Returns: The ConsumableBlocks

- **getEnabledState**
  Description: Gets the EnabledState of the memory as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState of the memory as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The EnabledState

- **getRequestedState**
  Description: Gets the last RequestedState of the memory
  Syntax: uint16 getRequestedState (void) const;
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the last RequestedState of the memory as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The RequestedState

- **getOperationalStatus**

Description: Gets the OperationalStatus of the memory
Syntax: vector<uint16> getOperationalStatus (void) const;
Returns: List of OperationalStatus

- **getOperationalStatusStr**
  Description: Gets the OperationalStatus of the memory as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: List of OperationalStatus

- **getHealthState**
  Description: Gets the HealthState of the memory
  Syntax: uint16 getHealthState (void) const;
  Returns: The HealthState

- **getHealthStateStr**
  Description: Gets the HealthState of the memory as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The HealthState

- **getElementName**
  Description: Gets the ElementName of the memory
  Syntax: string getElementName (void) const;
  Returns: The **ElementName**

Note: All the Member function in CPhysicalMemory throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- physicalmemory.h
Library -- dashapi

Usage Examples

Enumerate the physical memory and display the memory information

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "physicalmemory.h"
```

```
using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        /* enumerate the memory */
        CPhysicalMemory::iterator i = CPhysicalMemory::enumPhysicalMemory (client);
        for (; i != CPhysicalMemory::iterator::end (); ++i)
        {
                /* get the instances of the memory*/
                CPhysicalMemory pm = *i;

                try
                {
                        /* get the memory information */
                        pm.getSpeed ();
                        pm.getCapacity ();
                }
                catch (...)
                {
                        fprintf (stdout, "Error accessing physical memory\n");
                }
        }
        delete client;
        return 0;
}
```

3.3.5. CProcessorCore

This is a class that represents Processor Core.

Public Member Functions

- getCIMObject
- getInstanceID
- getCoreEnabledState
- getCoreEnabledStateStr
- getLoadPercentage
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr

- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getElementName

Static Member Functions

- **enumProcessorCores**
- **getCachedProps**

Constructors Description

- **CProcessorCore**

  Description: Constructs this object from the corresponding CIM_ProcessorCore object.

  Syntax: CProcessorCore (const CIM_ProcessorCore & pc);

  Parameters:
  - *pc* CIM Processor Core object.

Member Functions Description

- enumProcessorCores

  Description: Enumerates all the processor cores present under a Management Access Point.

  Syntax: CProcessorCore::iterator enumProcessorCores (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the processor core.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_ProcessorCore* getCIMObject (void);

  Returns: The underlying CIM object

- **getInstanceID**

  Description: Gets the instance ID of the processor Core

Syntax: string getInstanceID (void) const;
Returns: The Instance ID

- **getCoreEnabledState**
  Description: Gets the Core Enabled state of processor Core
  Syntax: uint16 getCoreEnabledState (void) const;
  Returns: The Core Enabled state

- **getLoadPercentage**
  Description: Gets the load percentage of this processor core
  Syntax: uint16 getLoadPercentage (void);
  Returns: The load percentage of this processor core.

- **getEnabledState**
  Description: Gets the state of the processor
  Syntax: uint16 getEnabledState (void) const;
  Returns: The enabled state

- **getEnabledStateStr**
  Description: Gets the state of the processor as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The enabled state

- **getRequestedState**
  Description: Gets the last requested state of the processor
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requested state

- **getRequestedStateStr**
  Description: Gets the last requested state of the processor as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The requested state

- **getOperationalStatus**
  Description: Gets the operational status of the processor
  Syntax: vector<uint16> getOperationalStatus (void) const;
  Returns: List of OperationalStatus

- **getOperationalStatusStr**
  Description: Gets the operational status of the processor as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: List of OperationalStatus

- **getHealthState**
  Description: Gets the health state of the processor
  Syntax: uint16 getHealthState (void) const;
  Returns: The health state

- **getHealthStateStr**
  Description: Gets the health state of the processor as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The health state

- **getElementName**
  Description: Gets the element name of the processor
  Syntax: string getElementName (void) const;
  Returns: The element name

  Note: All the Member function in CProcessorCore throws exceptions if error occurs. Following exceptions are thrown by this API.
- EDSDKError.
- ECIMError

  The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirement

Header file -- processor.h
Library -- dashapi

3.3.6. CProcessor

This class represents a Processor.

Member Functions

- getCIMObject
- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getFamily
- getCurrentClockSpeed
- getMaxClockSpeed
- getExternalBusClockSpeed
- getCPUStatus
- getCPUStatusStr

- getLoadPercentage
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getElementName
- getOtherFamilyDescription
- enableProcessor
- disableProcessor
- resetProcessor

Static Member Functions

- enumProcessors
- getCachedProps

Constructor Descriptions

- **CProcessor**
  Description: Construct this object from the corresponding CIM_Processor object.
  Syntax: CProcessor (const CIM_Processor & processor);
  Parameters:
  - *processor* CIM Processor object.

Member Functions Description

- **enumProcessors**
  Description: Enumerates all the processors present under a Management Access Point.
  Syntax: CProcessor::iterator enumProcessors (IClient* client,bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the processor.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_Processor* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**
  Description: Gets the processor SystemCreationClassName
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The processor SystemCreationClassName

- **getSystemName**
  Description: Gets the processor SystemName
  Syntax: string getSystemName (void) const;
  Returns: The processor SystemName

- **getCreationClassName**
  Description: Gets the processorCreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The processor CreationClassName

- **getDeviceID**
  Description: Gets the processor Device ID
  Syntax: string getDeviceID (void) const;
  Returns: The processor Device ID

- **getFamily**
  Description: Gets the processor family
  Syntax: string getFamily (void) ;
  Returns: The processor family.

- **getCurrentClockSpeed**
  Description: Gets the current clock speed
  Syntax: uint32 getCurrentClockSpeed (void);
  Returns: The current clock speed in Mhz

- **getMaxClockSpeed**
  Description: Gets the maximum clock speed.
  Syntax: uint32 getMaxClockSpeed (void);
  Returns: The maximum clock speed in Mhz.

- **getExternalBusClockSpeed**

Description: Gets the external bus clock speed
Syntax: uint32 getExternalBusClockSpeed (void);
Returns: The external bus clock speed.

- **getCPUStatus**
  Description: Gets the current status of the processor
  Syntax: uint16 getCPUStatus (void) const;
  Returns: The current status

- **getCPUStatusStr**
  Description: Gets the current status of the processor as string
  Syntax: string getCPUStatusStr (void) const;
  Returns: The current status

- **getLoadPercentage**
  Description: Gets the load of the processor in the last minute
  Syntax: uint32 getLoadPercentage (void);
  Returns: The load percentage

- **getEnabledState**
  Description: Gets the EnabledState of the processor
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState of the processor as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The EnabledState string

- **getRequestedState**
  Description: Gets the last RequestedState of the processor
  Syntax: uint16 getRequestedState (void) const;
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the last RequestedState of the processor as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The RequestedState string

- **getOperationalStatus**
  Description: Gets the OperationalStatus of the processor

Syntax: vector<uint16> getOperationalStatus (void) const;
Returns: List of OperationalStatus array

- **getOperationalStatusStr**
  Description: Gets the OperationalStatus of the processor as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: List of OperationalStatus string array

- **getHealthState**
  Description: Gets the HealthState of the processor
  Syntax: uint16 getHealthState (void) const;
  Returns: The HealthState

- **getHealthStateStr**
  Description: Gets the HealthState of the processor as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The HealthState string

- **getElementName**
  Description: Gets the ElementName of the processor
  Syntax: string getElementName (void) const;
  Returns: The ElementName

- **getOtherFamilyDescription**
  Description: Gets the family description if the family type is "other".
  Syntax: string getOtherFamilyDescription (void) const;
  Returns: The family description

- **enableProcessor**
  Description: Enables the processor
  Syntax: uint32 enableProcessor (void);
  Returns: 0 on success, throws execption on failure.

- **disableProcessor**
  Description: Disables the processor
  Syntax: uint32 disableProcessor (void);
  Returns: 0 on success, throws execption on failure.

- **resetProcessor**
  Description: Resets the processor
  Syntax: uint32 resetProcessor (void);
  Returns: 0 on success, throws execption on failure.

Note: All the Member function in CProcessor throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirement

Header file -- processor.h
Library -- dashapi

Usage Examples

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "processor.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        /* enumerate the processor */
        CProcessor::iterator i = CProcessor::enumProcessors (client);
        for (; i != CProcessor::iterator::end (); ++i)
        {
                /* get the instances of the asset*/
                CProcessor processor = *i;

                try
                {
                        /* get the processor information */
                        fprintf (stdout, "Processor status = %d\n", processor.getCPUStatus
                                                                                ());;
                        fprintf (stdout, "Processor family = %s\n", processor.getFamily
                                                                ().c_str ());
                }
                catch (...)
                {
                        fprintf (stdout, "Error accessing processor\n");
                }
```

```
        }
        delete client;

        return 0;
}
```

3.3.7. CPowerSupply

This is a class representing a Power Supply.

Member Functions

- getActiveInputVoltage
- getCIMObject
- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getTotalPower
- getElementName
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- enablePowerSupply
- disablePowerSupply
- resetPowerSupply
- offlinePowerSupply

Static Member Functions

- enumPowerSupplies
- getCachedProps

Constructor Description

- **CPowerSupply**
  Description: Construct this object from the corresponding CIM_PowerSyupply object.
  Syntax: CPowerSupply (const CIM_PowerSupply & ps);
  Parameters:

- *ps* CIM Power Supply object.

Member Functions Description

- **enumPowerSupplies**

  Description: Enumerates all the power supplies present under a management access point.

  Syntax: CPowerSupply::iterator enumPowerSupplies (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the power supply

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_PowerSupply* getCIMObject (void);

  Returns: The underlying CIM object

- **getSystemCreationClassName**

  Description: Gets the power supply System Creation class Name

  Syntax: string getSystemCreationClassName (void) const;

  Returns: The power supply system creation class name

- **getSystemName**

  Description: Gets the power supply System Name

  Syntax: string getSystemName (void) const;

  Returns: The power supply system name

- **getCreationClassName**

  Description: Gets the power supply Creation class Name

  Syntax: string getCreationClassName (void) const;

  Returns: The power supply creation class name

- **getDeviceID**

  Description: Gets the Device ID of the power supply system

  Syntax: string getDeviceID(void) const;

Returns: The DeviceID.

- **getTotalPower**
  Description: Gets the total power
  Syntax: uint16 getTotalPower (void);
  Returns: The total power.

- **getElementName**
  Description: Gets the Element Name of the power supply system
  Syntax: string getElementName(void) const;
  Returns: The ElementName.

- **getOperationalStatus**
  Description: Gets the operational status of the power supply
  Syntax: vector<uint16> getOperationalStatus (void) const;
  Returns: List of OperationalStatus string array

- **getOperationalStatusStr**
  Description: Gets the operational status of the power supply as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: List of OperationalStatus string array

- **getHealthState**
  Description: Gets the health state of the power supply
  Syntax: uint16 getHealthState (void) const;
  Returns: The health state

- **getHealthStateStr**
  Description: Gets the health state of the power supply as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The health state string

- **getEnabledState**
  Description: Gets the state of the power supply
  Syntax: uint16 getEnabledState (void) const;
  Returns: The enabled state

- **getEnabledStateStr**
  Description: Gets the state of the power supply as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The enabled state string

- **getRequestedState**

Description: Gets the last requested state of the power supply
Syntax: uint16 getRequestedState (void) const;
Returns: The requested state

- **getRequestedStateStr**
  Description: Gets the last requested state of the power supply as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The requested state string

- **enablePowerSupply**
  Description: Enables the power supply
  Syntax: uint32 enablePowerSupply (void);
  Returns: 0 on success, throws execption on failure.

- **disablePowerSupply**
  Description: Disables the power supply
  Syntax: uint32 disablePowerSupply (void);
  Returns: 0 on success, throws execption on failure.

- **resetPowerSupply**
  Description: Resets the power supply
  Syntax: uint32 resetPowerSupply (void);
  Returns: 0 on success, throws execption on failure.

- **offlinePowerSupply**
  Description: Makes the power supply offline.
  Syntax: uint32 offlinePowerSupply (void);
  Returns: 0 on success, throws execption on failure.

Note: All the Member function in CPowerSupply throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- powersupply.h
Library -- dashapi

3.3.8. CSensor

A class representing a Sensor.

Member Functions

- getCIMObject
- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getSensorType
- getSensorTypeStr
- getPossibleStates
- getCurrentState
- getElementName
- getOtherSensorTypeDescription
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr

Static Member Functions

- enumSensors
- getCachedProps

Constructor & Destructor Description

- **CSensor**

  Description: Construct this object from the corresponding CIM_Sensor object.

  Syntax: CSensor (const CIM_Sensor& Sensor);

  Parameters:

  - *sensor* CIM Sensor object.

Member Functions Description

- **enumSensors**

  Description: Enumerates all the senosrs present under a management access point.

  Syntax: CSensor::iterator enumSensors (IClient* client,
  bool cached = true);

  Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching. Default is true.

Returns: Iterator to the sensor.

- **getCachedProps**

  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object
  Syntax: CIM_Sensor* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**

  Description: Gets the System Creation class name of the sensor
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The System Creation Class name

- **getSystemName**

  Description: Gets the System nameof the sensor
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The System name

- **getSystemCreationClassName**

  Description: Gets the Creation class name of the sensor
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The Creation Class name

- **getDeviceID**

  Description: Gets the device id of the sensor
  Syntax: string getDeviceID (void) const;
  Returns: The sensor device id.

- **getSensorType**

  Description: Gets the type of the sensor
  Syntax: uint16 getSensorType (void) const;
  Returns: The sensor type.

- **getSensorTypeStr**

  Description: Gets the type of the sensor as string
  Syntax: string getSensorTypeStr (void) const;

Returns: The sensor type string.

- **getPossibleStates**
  Description: Gets the possible states of the sensor.
  Syntax: vector<string> getPossibleStates (void) const;
  Returns: List of possible states of the sensor.

- **getCurrentState**
  Description: Gets the current state of the sensor.
  Syntax: string getCurrentState (void) const;
  Returns: The current state of the sensor.

- **getElementName**
  Description: Gets the element name of the sensor
  Syntax: string getElementName (void) const;
  Returns: The element name

- **getOtherSensorTypeDescription**
  Description: Gets the sensor type description if the sensor type is "other".
  Syntax: string getOtherSensorTypeDescription (void) const;
  Returns: The sensor type description

- **getEnabledState**
  Description: Gets the state of the sensor
  Syntax: uint16 getEnabledState (void) const;
  Returns: The enabled state

- **getEnabledStateStr**
  Description: Gets the state of the sensor as sting
  Syntax: string getEnabledStateStr (void) const;
  Returns: The enabled state string

- **getRequestedState**
  Description: Gets the Requested state of the sensor
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requested state

- **getRequestedStateStr**
  Description: Gets the Requested state of the sensor as sting
  Syntax: string getRequestedStateStr (void) const;
  Returns: The requested state string

- **getOperationalStatus**

Description: Gets the operational status of the sensor
Syntax: vector<uint16> getOperationalStatus (void) const;
Returns: List of operational status array

- **getOperationalStatusStr**
  Description: Gets the operational status of the sensor as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: List of OperationalStatus string

- **getHealthState**
  Description: Gets the health state of the sensor
  Syntax: uint16 getHealthState (void) const;
  Returns: The health state

- **getHealthStateStr**
  Description: Gets the health state of the sensor as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The health state string

Note: All the Member function in CSensor throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- sensor.h
Library -- dashapi

3.3.9. CSoftware

This a class that represents Software.

Public Functions

- getCachedProps
- getCIMObject
- getInstanceID
- getIsEntity
- getVersionString
- getMajorVersion
- getMinorVersion
- getRevisionNumber

- getBuildNumber
- getTargetOSTypes
- getTargetOperatingSystems
- getIdentityInfoType
- getIdentityInfoValue
- getClassifications
- getClassificationsStr

Static Member Functions

- enumSoftware
- getCachedProps

Constructor Description

- **CSoftware**

  Description: Constructs this object from the corresponding CIM_ SoftwareIdentity object.

  Syntax: CSoftware (const CIM_SoftwareIdentity& software);

  Parameters:

  - *software* CIM Software Identity object.

Member Functions Description

- **enumSoftware**

  Description: Enumerates all the software present under a management access point.

  Syntax: CSoftware::iterator enumSoftware (IClient* client, bool cached = true);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the software.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_SoftwareIdentity* getCIMObject (void);

  Returns: The underlying CIM object

- **getInstanceID**
  Description: Gets the instance id of the software
  Syntax: string getInstanceID (void);
  Returns: The instance id

- **getIsIdentity**
  Description: Gets IsEntity - whether the SoftwareIdentity corresponds to a discrete copy of the software component or is being used to convey descriptive and identifying information about software that is not present in the management domain. A value of TRUE shall indicate that the SoftwareIdentity instance corresponds to a discrete copy of the software component. A value of FALSE shall indicate that the SoftwareIdentity instance does not correspond to a discrete copy of the Software.
  Syntax: boolean getIsEntity (void) const;
  Returns: the boolean value 0 if true else false

- **getVersionString**
  Description: Gets the version string
  Syntax: string CSoftware::getVersionString (void);
  Returns: The version string.

- **getMajorVersion**
  Description: Gets the major version
  Syntax: uint16 getMajorVersion (void);
  Returns: The major version.

- **getMinorVersion**
  Description: Gets the minor version
  Syntax: uint16 getMinorVersion (void);
  Returns: The minor version

- **getRevisionNumber**
  Description: Gets the revision number
  Syntax: uint16 getRevisionNumber (void);
  Returns: The revision number

- **getBuildNumber**
  Description: Gets the build number
  Syntax: uint16 getBuildNumber (void);
  Returns: The build number

- **getTargetOSTypes**
  Description: Gets an array of TargetOSTypes
  Syntax: vector<uint16> getTargetOSTypes (void) const;
  Returns: The list of target OS types

- **getTargetOperatingSystems**
  Description: Gets the list of target operating systems
  Syntax: vector<string> getTargetOperatingSystems (void);
  Returns: The list of target OS

- **getIdentityInfoType**
  Description: Gets an array of IdentityInfoType
  Syntax: vector<string> getIdentityInfoType (void) const;
  Returns: The type of information

- **getIdentityInfoValue**
  Description: Gets an array of IdentityInfoValue
  Syntax: vector<string> getIdentityInfoValue (void) const;
  Returns: The identify a software instance within the context of the organization. For example, large organizations may have several ways to address or identify a particular instance of software depending on where it is stored; a catalog, a web site, or for whom it is intended; development, customer service, etc. The indexed array property IdentityInfoValue contains 0 or more strings that contain a specific identity info string value. IdentityInfoValue is mapped and indexed to IdentityInfoType. When the IdentityInfoValue property is implemented, the IdentityInfoType property MUST be implemented and shall be formatted using the algorithm provided in the IdentityInfoType property Description.

- **getClassifications**
  Description: Gets the classification
  Syntax: vector<uint16> getClassifications (void);
  Returns: The list of classification

- **getClassificationsStr**
  Description: Gets the classification as string
  Syntax: vector<string> getClassificationsStr (void) const;
  Returns: The list of classification as string

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

- EFunctionNotSupported
- EfunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- software.h
Library -- dashapi

3.3.10. CBootConfig

This a class that represents a Boot configuration. It includes Boot Device class also.

Member Functions for BootDevice

- getInstanceID
- getElementName
- getBootString
- getBIOSBootString
- getStructuredBootString
- getFailThroughSupported
- getFailThroughSupportedStr

BootDevice Static Member Functions

- enumBootDevices
- getCachedProps
- compareBootOrder

Member Functions for BootConfig

- getInstanceID
- getElementName
- isDefaultBoot
- isCurrentBoot
- isNextBoot
- setDefaultBoot
- setNextBoot
- getBootOrder
- changeBootOrder
- deleteBootConfig
- capableOfBootConfigManagement
- capableOfAddBootconfig
- capableOfDeleteBootConfig

- capableOfSetDefaultBoot
- capableOfChangeBootOrder
- getBootStringsSupported
- getBootStringsSupportedStr
- getBootCapabilitesSupported
- getBootCapabilitesSupportedStr

BootConfig Static Member Functions

- enumBootConfigs
- getCachedProps
- addBootConfig

Constructor Descriptions

- **CBootDevice**

  Description: Construct this object from the corresponding CIM_BootSourceSetting object.

  Syntax: CBootDevice (const CBootDevice& bd);

  Parameters:
  - *bd* CIM Boot Source Setting object

- **CBootConfig**

  Description: Construct this object from the corresponding CIM_BootConfigSetting object.

  Syntax: CBootConfig (const CIM_BootConfig& bc);

  Parameters:
  - *bc* CIM Boot Config object.

Member Functions Description for BootDevice

- **enumBootDevices**

  Description: Enumerates all the boot devices present under a management access point.

  Syntax: CBootDevice::iterator enumBootDevices (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the Boot device.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

Returns: List of cached properties.

- **compareBootOrder**
  Description: Compare the boot order of the Boot Devices
  Syntax: static bool compareBootOrder (const CBootDevice& first, const CBootDevice& second);
  Parameters:
  - first First boot device
  - second Second boot device

  Returns: 0 if same else false

- **getInstanceID**
  Description: Gets the instance id
  Syntax: string getInstanceID (void) const;
  Returns: The instance id

- **getElementName**
  Description: Gets the element name
  Syntax: string getElementName (void) const;
  Returns: The element name

- **getBootString**
  Description: Gets the Boot String
  Syntax: string getBootString (void) const;
  Returns: The boot string

- **getBIOSBootString**
  Description: Gets the BIOS Boot String
  Syntax: string getBIOSBootString (void) const;
  Returns: The BIOS boot string

- **getStructuredBootString**
  Description: Gets the Structured Boot String
  Syntax: string getStructuredBootString (void) const;
  Returns: The structured boot string

- **getFailThroughSupported**
  Description: Gets the Fail through supported
  Syntax: uint16 getFailThroughSupported (void) const;
  Returns: The fail through supported

- **getFailThroughSupportedStr**
  Description: Gets the Fail through supported as string

Syntax: string getFailThroughSupportedStr (void) const;
Returns: The fail through supported

Member Functions Description for BootConfig

- **enumBootConfigs**
  Description: Enumerates all the boot configuration present under a management access point.
  Syntax: CBootConfigs::iterator enumBootConfigs (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the Boot config.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_BootConfigSetting* getCIMObject (void);
  Returns: The underlying CIM object
- **addBootConfig**
  Description: Adds a boot configuration to the computer system specified in cs.
  Syntax: CBootConfig add (CComputerSystem& *cs*, CBootConfig& bc);
  Parameters:
  - *cs* Computer system where the new config to be added
  - bc Existing boot configuration used as template

  Returns: The added boot configuration.

- **getInstanceID**
  Description: Gets the instance id of this boot configuration.
  Syntax: string getInstanceID (void);
  Returns: The instance id.

- **getElementName**
  Description: Gets the element name of this boot configuration.
  Syntax: string getElementName (void);
  Returns: The element name

.
- **isDefaultBoot**

Description: Checks the system and returns.
Syntax: int isDefaultBoot (void);
Returns: 1 if this is default boot configuration.

- **isCurrentBoot**
  Description: Checks the system and returns.
  Syntax: int isCurrentBoot (void);
  Returns: 1 if this is current boot configuration.

- **isNextBoot**
  Description: Checks the system and returns.
  Syntax: int isNextBoot (void);
  Returns: 1 if this is next boot configuration.

- **setDefaultBoot**
  Description: Sets this boot configuration as default boot configuration.
  Syntax: uint32 setDefaultBoot (void);
  Returns: 0 if success else the error code.

- **setNextBoot**
  Description: Sets this boot configuration as next boot configuration.
  Syntax: uint32 setNextBoot (void);
  Returns: 0 if success else the error code.

- **getBootOrder**
  Description: Gets the boot order from the .boot configuration.
  Syntax: vector<BootDeviceInfo_T> (void);
  Returns: The device boot order for this configuration.

- **changeBootOrder**
  Description: Changes the boot device order of the boot configuration.
  Syntax: uint32 changeBootOrder (vector<BootDeviceInfo_Tboot_order);
  Parameters:
    - boot_order The new boot device order.
  Returns: 0 if boot order changed successfully, else will return the error code.

- **deleteBootConfig**
  Description: Deletes this boot configuration.
  Syntax: void delete (void);

- **capableOfBootConfigManagement**

Description: Verifies whether the BootService and BootServiceCapabilities instances are *exists or not*.

Syntax: ***bool capableOfBootConfigManagement(const CComputerSystem &cs);***

Parameters:

- ***CComputerSystem*** The CComputerSystem instance.

Returns: True or False

- **capableOfBootConfigManagement**

Description: Verifies whether the BootService and BootServiceCapabilities instances are *exists or not*.

Syntax: ***bool capableOfBootConfigManagement(const CComputerSystem &cs, CIM_BootService &bs, CIM_BootServiceCapabilities &bsc);***

Parameters:

- ***CComputerSystem*** The CComputerSystem instance.
- ***CIM_BootService*** Output parameter passed as reference
- ***CIM_BootServiceCapabilities*** Output parameter passed as reference

Returns: True or False

- **capableOfAddBootconfig**

Description: Verifies whether AddBootConfig operation can be performed or not.

Syntax: ***bool capableOfAddBootconfig(const CComputerSystem &cs);***

Parameters:

- ***CComputerSystem*** The CComputerSystem instance.

Returns: True or False

- **capableOfDeleteBootconfig**

Description: Verifies whether DeleteBootConfig operation can be performed or not.

Syntax: ***bool capableOfDeleteBootconfig(const CComputerSystem &cs);***

Parameters:

- ***CComputerSystem*** The CComputerSystem instance.

Returns: True or False

- **capableOfSetDefaultBoot**

Description: Verifies whether SetDefaultBoot operation can be performed or not.

Syntax: ***bool capableOfSetDefaultBoot(CIM_BootService &bs);***

Parameters:

- ***CIM_BootService*** Output parameter passed as reference

Returns: True or False

- **capableOfSetDefaultBoot**

Description: Verifies whether the SetDefaultBoot operation can be performed or not.

Syntax: ***bool capableOfSetDefaultBoot(void);***

Returns: True or False

- **capableOfChangeBootOrder**
  Description: Verifies whether ChangeBootOrder operation can be performed or not.
  Syntax: *bool* ***capableOfChangeBootOrder(void);***
  Returns: True or False

- **getBootStringsSupported**
  Description: Get the BootStringsSupported values as integer.
  Syntax: *bool* ***getBootStringsSupported(****const CComputerSystem &cs , vector<uint16> &val);***
  Parameters:
  - ***CComputerSystem*** The CComputerSystem instance.
  - ***Vector<uint16>*** Integer vector reference as out paratmeter.

  Returns: True or False

- **getBootStringsSupportedStr**
  Description: Get the BootStringsSupported values as string.
  Syntax: *bool* ***getBootStringsSupported(****const CComputerSystem &cs , vector<string> &str);***
  Parameters:
  - ***CComputerSystem*** The CComputerSystem instance.
  - ***Vector<string>*** String vector reference as out paratmeter.

  Returns: True or False

- **getBootCapabilitesSupported**
  Description: Get the BootCapabilitiesSupported values as integer
  Syntax: *bool* ***getBootCapabilitiesSupported(****const CComputerSystem &cs , vector<uint16> &val);***
  Parameters:
  - ***CComputerSystem*** The CComputerSystem instance.
  - ***Vector<uint16>*** Integer vector reference as out paratmeter.

  Returns: True or False

- **getBootCapabilitesSupportedStr**
  Description: Get the BootCapabilitiesSupported values as string
  Syntax: *bool* ***getBootCapabilitiesSupported(****const CComputerSystem &cs , vector<string> &str);***
  Parameters:
  - ***CComputerSystem*** The CComputerSystem instance.
  - ***Vector<string>*** String vector reference as out paratmeter.

Returns: True or False

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- bootconfig.h
Library -- dashapi

3.3.11. CUser

A class representing a User account. It includes CRole class also.

Member Functions for Role

- getCreationClassName
- getName
- getRoleCharacteristics
- getCommonName
- getElementName
- deleteRole
- modifyRole
- assignPermissions
- getPermissions
- capableOfRoleBasedAuthorizationService
- capableOfCreateRole
- capableOfDeleteRole
- capableOfModifyRole
- capableOfAssignRoles
- capableOfShowRoles
- capableOfShowAccess
- isMethodSupported
- getSupportedMethodsStr

Static Member Functions for Role

- enumRoles
- createRole

- getSupportedActivityQualifiers
- getCachedProps

Member Functions for User

- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getUserID
- getUserPassword
- getOrganizationName
- getElementName
- getUserPasswordEncryptionAlgorithm
- getOtherUserPasswordEncryptionAlgorithm
- getPasswordHistoryDepth
- getPasswordExpiration
- getComplexPasswordRulesEnforced
- getInactivityTimeout
- getMaximumSuccessiveLoginFailures
- getRequestedState
- getRequestedStateStr
- getEnabledState
- getEnabledStateStr
- getUserRoles
- assignRoles
- removeRoles
- deleteUser
- enableUser
- disableUser
- capableOfAccountManagementService
- isMethodSupported
- changePassword
- capableOfCreatUser
- capableOfDeleteUser
- capableOfModifyUser
- capableOfRoleBasedAuthorizationService
- capableOfAssignRoles
- capableOfRemoveRoles
- capableOfRequestStateChange
- getRequestedStatesSupported

Static Member Functions for User

- enumUsers
- createUser
- getCachedProps

Constructor & Destructor Descriptions

- CUser
  Description: Construct this object from the corresponding CIM_Account object.
  Syntax: CUser (const CIM_Account& user);
  Parameters:
  - *user* CIM account object.

- **CRole**
  Description: Construct this object from the corresponding CIM_Role object.
  Syntax: CRole (const CIM_Role& role);
  Parameters:
  - role CIM role object

Member Functions Description for Role

- **enumRoles**
  Description: Enumerates all the roles present under a management access point.
  Syntax: CRole::iterator enumRoles (IClient* client,
  bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the user
- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_Role* getCIMObject (void);
  Returns: The underlying CIM object

- **getCreationClassName**
  Description: Gets the CreationClassName of this Role

Syntax: string getCreationClassName (void) const;
Returns: The CreationClassName

- **getName**
  Description: Gets name for this Role
  Syntax: string getName (void);
  Returns: The name

- **getRoleCharacteristics**
  Description: Gets the RoleCharacteristics of this Role
  Syntax: string getRoleCharacteristics (void)
  Returns: The RoleCharacteristics
- **getCommonName**
  Description: Gets the CommonName of this Role
  Syntax: string getCommonName (void) const;
  Returns: The CommonName

- **getElementName**
  Description: Gets the ElementName of this Role
  Syntax: string getElementName (void) const;
  Returns: The ElementName.

- **deleteRole**
  Description: Deletes the Role
  Syntax: void deleteRole (void);
  Returns: None

- **createRole**
  Description: Creats a new role with the permissions specified
  Syntax: static CRole createRole (const CcomputerSystem& cs, const string& name, const vector<CRole::Permission_T>& permissions);
  Parameters:
  - cs Computer System where the role will be added.
  - Name Name of the role
  - permissions Privileges/permissions for this role(Permission_T (activities, qualifiers & formats)
    Permissions are created using the
    1. activities (Activity_E)
    (Create|Delete|Detect|Read|Write| Execute|Other).
    2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
    3. and formats (QualifierFormats_E), this is optional(targets may or may not support this).

  Returns: none

- **modifyRole**

  Description: Modify this role with new privileges/permissons, this will replace the existing permission with new permissions

  Syntax: void modifyRole (const vector<CRole::Permission_T>& permissions);

  parameters:

  - permissions Privileges/permissions for this role(Permission_T (activities, qualifiers & formats))
    Permissions are created using the
    1. activities (Activity_E) (Create|Delete|Detect|Read|Write| Execute|Other).
    2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
    3. and formats (QualifierFormats_E), this is optional(targets may or may Not support this).

  Returns: none

- **assignPermissions**

  Description: Assigns the Role

  Syntax: void assignPermissions (vector<CRole::Permission_T> permissions);

  Parameters:

  - permissions Privileges/permissions for this role(Permission_T (activities, qualifiers & formats))
    Permissions are created using the
    1. activities (Activity_E) (Create|Delete|Detect|Read|Write| Execute|Other).
    2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
    3. and formats (QualifierFormats_E), this is optional(targets may or may not support this).

  Returns: none

- **getPermissions**

  Description: Gets the permissions for this role

  Syntax: vector<CRole::Permission_T> getPermissions (void) const;

  Returns: The permissions for this role

- **getSupportedActivityQualifiers**

  Description: Gets the activity qualifiers supported by the target/MAP.

  Syntax: static vector<string> getSupportedActivityQualifiers (IClient* client);

  Parameters:

  - client Pointer to the client interface.

  Returns: The activity qualifiers supported by the target/MAP.

- **capableOfRoleBasedAuthorizationService**

  Description: Verifies whether the RoleBasedAuthorization Service and Capabilitiies instances are exists or not.

  Syntax: ***bool capableOfRoleBasedAuthorizationService (void);***

  Returns: True or False

- **capableOfRoleBasedAuthorizationService**

  Description: Verifies whether the RoleBasedAuthorization Service and Capabilitiies instances are exists or not.

  Syntax: *bool* ***capableOfRoleBasedAuthorizationService*** *(CIM_ RoleBasedAuthorizationService &rbas, CIM_ RoleBasedManagemetnCapabilities &rbmc);*

  parameters:

  - *CIM_ RoleBasedAuthorizationService* Output parameter passed as reference
  - *CIM_ RoleBasedManagemetnCapabilities* Output parameter passed as reference

  Returns: True or False

- **capableOfCreateRole**

  Description: Verifies whether the CreateRole operations can be performed or not.

  Syntax: ***bool capableOfCreateRole()***

  Returns: True or False

- **capableOfCreateRole**

  Description: Verifies whether the CreateRole operations can be performed or not.

  Syntax: ***bool capableOfCreateRole (****CIM_RoleBasedAuthorizationService &rbas****);***

  parameters:

  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfDeleteRole**

  Description: Verifies whether the DeleteRole operations can be performed or not.

  Syntax: ***bool capableOfDeleteRole()***

  Returns: True or False

- **capableOfDeleteRole**

  Description: Verifies whether the DeleteRole operations can be performed or not.

  Syntax: ***bool capableOfDeleteRole(****CIM_RoleBasedAuthorizationService &rbas****);***

  parameters:

  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

Returns: True or False

- **capableOfModifyRole**
  Description: Verifies whether the ModifyRole operations can be performed or not.
  Syntax: ***bool capableOfModifyRole()***
  Returns: True or False

- **capableOfModifyRole**
  Description: Verifies whether the ModifyRole operations can be performed or not.
  Syntax: ***bool capableOfModifyRole(****CIM_RoleBasedAuthorizationService &rbas****);***
  parameters:
  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfAssignRoles**
  Description: Verifies whether the AssignRoles operations can be performed or not.
  Syntax: ***bool capableOfAssignRoles()***
  Returns: True or False

- **capableOfAssignRoles**
  Description: Verifies whether the AssignRoles operations can be performed or not.
  Syntax: ***bool capableOfAssignRoles (****CIM_RoleBasedAuthorizationService &rbas****);***
  parameters:
  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfShowRoles**
  Description: Verifies whether the ShowRoles operations can be performed or not.
  Syntax: ***bool capableOfShowRoles()***
  Returns: True or False

- **capableOfShowRoles**
  Description: Verifies whether the ShowRoles operations can be performed or not.
  Syntax: ***bool capableOfShowRoles (****CIM_RoleBasedAuthorizationService &rbas****);***
  parameters:
  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

Returns: True or False

- **capableOfShowAccess**

  Description: Verifies whether the ShowAccess operations can be performed or not.

  Syntax: ***bool capableOfShowAccess (****CIM_RoleBasedAuthorizationService &rbas****);***

  parameters:

  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfShowAccess**

  Description: Verifies whether the ShowAccess operations can be performed or not.

  Syntax: ***bool capableOfShowAccess();***

  Returns: True or False

- **isMethodSupported**

  Description: Verifies whether the particular MethodSupported values propery exists or not.

  Syntax: ***bool isMethodSupported(****CIM_RoleBasedAuthorizationService &rbas ,uint16 val****);***

  parameters:

  - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference
  - *Uint16 val* Interger value

  Returns: True or False

- **getSupportedMethodsStr**

  Description: Get the SupportedMethods values as string vector

  Syntax: ***bool getSupportedMethodsStr(****vector<string> str****);***

  parameters:

  - *vector<string> &str* Output parameter vector<string> passed as reference

  Returns: True or False

Member Functions Description for User

- **enumUsers**

  Description: Enumerates all the users present under a management access point.

  Syntax: CUser::iterator enumUsers (IClient* client,
  bool cached = true);

  Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching. Default is true.

Returns: Iterator to the user

- **getCachedProps**

  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object
  Syntax: CIM_User* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**

  Description: Gets SystemCreationClassName for user
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The systemcreationclassname

- **getCreationClassName**

  Description: Gets CreationClassName for user
  Syntax: string getCreationClassName (void) const;
  Returns: The creationclassname
- **getSystemName**

  Description: Gets SystemName for user
  Syntax: string getSystemName (void) const;
  Returns: The systemname

- **getName**

  Description: Gets Name for user
  Syntax: string getName (void) const;
  Returns: The name

- **getUserID**

  Description: Gets UserID
  Syntax: string getUserID (void) const;
  Returns: The User ID

- **getUserPassword**

  Description: Gets an array of UserPassword
  Syntax: vector<string> getUserPassword (void) const;
  Returns: The password

- **getOrganizationName**
  Description: Gets an array of OrganizationName
  Syntax: vector<string> getOrganizationName (void) const;
  Returns: The organizationname

- **getElementName**
  Description**:** Gets ElementName
  Syntax: string getElementName (void) const;
  Returns: The ElementName

- **getUserPasswordEncryptionAlgorithm**
  Description: Gets UserPasswordEncryptionAlgorithm
  Syntax: uint16 getUserPasswordEncryptionAlgorithm (void) const;
  Returns: The userpasswordencryptionalogrithm

- **getOtherUserPasswordEncryptionAlgorithm**
  Description: Gets OtherUserPasswordEncryptionAlgorithm
  Syntax: string getOtherUserPasswordEncryptionAlgorithm (void) const;
  Returns: The otheruserpasswordencryptionalgorithm

- **getPasswordHistoryDepth**
  Description: Gets PasswordHistoryDepth
  Syntax: uint16 getPasswordHistoryDepth (void) const;
  Returns: The passwordhistorydepth

- **getPasswordExpiration**
  Description: Gets PasswordExpiration
  Syntax: datetime getPasswordExpiration (void) const;
  Returns: The passwordexpiration

- **getComplexPasswordRulesEnforced**
  Description: Gets an array of ComplexPasswordRulesEnforced
  Syntax: vector<uint16> getComplexPasswordRulesEnforced (void) const;
  Returns: The list of complexpasswordrulesenforced

- **getInactivityTimeout**
  Description: Gets InactivityTimeout
  Syntax: datetime getInactivityTimeout (void) const;
  Returns: The inactivity timeout

- **getMaximumSuccessiveLoginFailures**
  Description: Gets MaximumSuccessiveLoginFailures
  Syntax: uint16 getMaximumSuccessiveLoginFailures (void) const;

Returns: The maximumsuccessiveloginfailures

- **getRequestedState**
  Description: Gets RequestedState
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requestedstate

- **getRequestedStateStr**
  Description: Gets the last RequestedState of the user as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The RequestedState string
- **getEnabledState**
  Description: Gets EnabledState
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState of the user as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The EnabledState string

- **createUser**
  Description: Creats a user
  Syntax:
```
static CUser createUser (const CComputerSystem& cs,
                         const string&user_name,
                         const string&password,
                         const string&organizationname = "");
```
  Parameters:
    - cs Computer system where the user will be added.
    - user_name User name.
    - password User password.
    - organizationname optional Organization Name

  Returns: none

- **getUserRoles**
  Description: Gets the associated role with this user
  Syntax: vector<CRole> getUserRoles (void) const;
  Returns: The user roles

- **assignRoles**
  Description: Assign Role(s) to this user.
  Syntax: void assignRoles (const vector<string>& roles);
  parameters:

- roles Role names to assign for this user.

Returns: None

- **removeRoles**

  Description: Remove Role(s) from this user.

  Syntax: void removeRoles (const vector<string>& roles);

  parameters:

  - roles Role names to remove for this user.

  Returns: None

- **deleteUser**

  Description: Delets this user

  Syntax: void deleteUser (void);

  Returns: None

- **enableUser**

  Description: Enables User

  Syntax: uint32 enableUser (void);

  Returns: None

- **disableUser**

  Description: Disables User

  Syntax: uint32 disableUser (void);

  Returns: None

- **changePassword**

  Description: set password /Modify the password

  Syntax: void changePassword (string password);

  parameters:

  - password Password to change

  Returns: None

- **capableOfAccountManagementService**

  Description: Verifies whether the AccountManagement Service & capabilities instances are exists or not.

  Syntax: ***bool capableOfAccountManagementService(void);***

  Returns: True or False

- **capableOfAccountManagementService**

  Description: Verifies whether the AccountManagement Service & capabilities instances are exists or not.

  Syntax: *bool* ***capableOfAccountManagementService****(CComputerSystem cs, CIM_AccountManagementService &ams,*

*CIM_AccountManagementCapabilities &amc**);***

parameters:

- *CComputerSystem* Output parameter passed as reference
- *CIM_AccountManagementService* Output parameter passed as reference
- *CIM_AccountManagementCapabilities* Output parameter passed as reference

Returns: True or False

- **isMethodSupported**

Description: Verifies whether the perticulare method values exists or not.

Syntax: *bool **isMethodSupported**(CIM_AccountManagementCapabilities &amc, vector<uint16> val**);***

parameters:

- *CIM_AccountManagementCapabilities* Capabilitiy instance
- *Vector<uint16> val* Method value

Returns: True or False

- **capableOfCreatUser**

Description: Verifies whether the createUser operations can be performed or not.

Syntax: ***bool capableOfCreatUser()***

Returns: True or False

- **capableOfCreatUser**

Description: Verifies whether the createUser operations can be performed or not.

Syntax: ***bool capableOfCreatUser()*** *(CComputerSystem cs,*
CIM_AccountManagementService &ams,
*CIM_AccountManagementCapabilities &amc**);***

parameters:

- *CComputerSystem* Output parameter passed as reference
- *CIM_AccountManagementService* Output parameter passed as reference
- *CIM_AccountManagementCapabilities* Output parameter passed as reference

Returns: True or False

- **capableOfDeleteUser**

Description: Verifies whether the DeleteUser operations can be performed or not.

Syntax: ***bool capableOfDeleteUser()***

Returns: True or False

- **capableOfDeleteUser**

Description: Verifies whether the DeleteUser operations can be performed or not.

Syntax: ***bool capableOfDeleteUser()*** *(CComputerSystem cs,*
CIM_AccountManagementService &ams,
*CIM_AccountManagementCapabilities &amc)**;***

parameters:

- *CComputerSystem* Output parameter passed as reference
- *CIM_AccountManagementService* Output parameter passed as reference
- *CIM_AccountManagementCapabilities* Output parameter passed as reference

Returns: True or False

- **capableOfModifyUser**

Description: Verifies whether the ModifyUser operations can be performed or not.

Syntax: ***bool capableOfModifyUser()***

Returns: True or False

- **capableOfModifyUser**

Description: Verifies whether the ModifyUser operations can be performed or not.

Syntax: ***bool capableOfModifyUser()*** *(CComputerSystem cs,*
CIM_AccountManagementService &ams,
*CIM_AccountManagementCapabilities &amc)**;***

parameters:

- *CComputerSystem* Output parameter passed as reference
- *CIM_AccountManagementService* Output parameter passed as reference
- *CIM_AccountManagementCapabilities* Output parameter passed as reference

Returns: True or False

- **capableOfRoleBasedAuthorizationService**

Description: Verifies whether the RoleBasedAuthorization Service instances exists or not.

Syntax: ***bool capableOfRoleBasedAuthorizationService (void);***

Returns: True or False

- **capableOfRoleBasedAuthorizationService**

Description: Verifies whether the RoleBasedAuthorization Service instances exists or not.

Syntax: *bool* ***capableOfRoleBasedAuthorizationService*** *(CComputerSystem cs,*
CIM_ RoleBasedAuthorizationService &rbas);

parameters:

- *CIM_ RoleBasedAuthorizationService* Output parameter passed as reference

Returns: True or False

- **capableOfAssignRoles**
  Description: Verifies whether the AssignRoles operations can be performed or not.
  Syntax: ***bool capableOfAssignRoles()***
  Returns: True or False

- **capableOfAssignRoles**
  Description: Verifies whether the AssignRoles operations can be performed or not.
  Syntax: ***bool capableOfAssignRoles(****CIM_RoleBasedAuthorizationService &rbas****);***
  parameters:
    - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfRemoveRoles**
  Description: Verifies whether the RemoveRoles operations can be performed or not.
  Syntax: ***bool capableOfRemoveRoles()***
  Returns: True or False

- **capableOfRemoveRoles**
  Description: Verifies whether the RemoveRoles operations can be performed or not.
  Syntax: ***bool capableOfRemoveRoles(****CIM_RoleBasedAuthorizationService &rbas****);***
  parameters:
    - *CIM_RoleBasedAuthorizationService* Output parameter passed as reference

  Returns: True or False

- **capableOfRequestStateChange**
  Description: Verifies whether the RequestStateChange operations can be performed or not.
  Syntax: ***bool capableOfRequestStateChange(void);***
  Returns: True or False

- **getRequestedStatesSupported**
  Description: Get the RequestedStatesSupported string values.
  Syntax: ***bool getRequestedStatesSupported(vector <string> & str);***
  parameters:
    - *vector<string> & str* Output parameter passed as reference

  Returns: True or False

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionNotSupported
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- user.h
Library -- dashapi

3.3.12. CfanRedundancySet

A class representing a Fan Redundancy set.

Member Functions

- getInstanceID
- getRedundancyStatus
- getType
- getMinimumNumberNeeded
- getElementName
- failover

Static Member Functions

- enumFanRedundancySets
- getCachedProps

Member Enumeration Description

- **enum CFanRedundancySet::Type_E**
  Type of the redundancy set

Constructor & Desctructor Descriptions

- **CFanRedundancySet**
  Description: Construct this object from the corresponding CIM_RedundancySet object
  Syntax: CFanRedundancySet (const CIM_RedundancySet& rs);
  Parameters:
- *rs* CIM Redundancy set object;

Member Functions Description

- **enumRedundancySets**

  Description: Enumerates all the fan redundancy sets present under a management access point.

  Syntax: CFanRedundancySet::iterator enumRedundancySets (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the fan redundancy set.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getInstanceID**

  Description: Gets the InstanceID of redundancy sets supported

  Syntax: string getInstanceID (void);

  Returns: The Instance ID

- **getRedundancyStatus**

  Description: Gets the current redundancy status

  Syntax: int getRedundancyStatus (void);

  Returns: Returns the current redundancy status.

- **getType**

  Description: Gets the type of the redundancy set

  Syntax: vector<Type_E> getType (void) const;

  Returns: Returns the type of the redundancy set.

- **getMinimumNumberNeeded**

  Description: Gets the minimum number needed

  Syntax: uint32 getMinimumNumberNeeded (void) const;

  Returns: The minumum number
- **getElementName**

  Description: Gets the name of the Element

  Syntax: string getElementName(void) const;

  Returns: Returns the name of the Element

- **failover**

  Description: Forces a failover from one fan to another fan

  Syntax: uint32 failover (CFan& fan_from, CFan& fan_to);

Returns: Returns The status.

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- fan.h
Library -- dashapi

3.3.13. CPowerSupplyRedundancySet

A class representing a PowerSupply Redundancy set.

Member Functions

- getInstanceID
- getRedundancyStatus
- getType
- getMinimumNumberNeeded
- getElementName
- failover

Static Member Functions

- enumPowerSupplyRedundancySets
- getCachedProps

Member Enumeration Description

- **enum CPowerSupplyRedundancySet::Type_E**
  Type of the redundancy set

Constructor & Desctructor Descriptions

- **CPowerSupplyRedundancySet**
  Description: Construct this object from the corresponding CIM_RedundancySet object
  Syntax: CFanRedundancySet (const CIM_RedundancySet& rs);
  Parameters:
- *rs* CIM Redundancy set object;

Member Functions Description

- **enumRedundancySets**

  Description: Enumerates all the fan redundancy sets present under a management access point.

  Syntax: CPowerSupplyRedundancySet::iterator enumRedundancySets (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the PowerSupply redundancy set.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getInstanceID**

  Description: Gets the InstanceID of redundancy sets supported

  Syntax: string getInstanceID (void);

  Returns: The Instance ID

- **getRedundancyStatus**

  Description: Gets the current redundancy status

  Syntax: uint16 getRedundancyStatus (void) const;

  Returns: Returns the current redundancy status.

- **getType**

  Description: Gets the type of the redundancy set

  Syntax: vector<Type_E> getType (void) const;

  Returns: Returns the type of the redundancy set.

- **getMinimumNumberNeeded**

  Description: Gets the minimum number needed

  Syntax: uint32 getMinimumNumberNeeded (void) const;

  Returns: The minumum number

- **getElementName**

  Description: Gets the name of the Element

  Syntax: string getElementName(void) const;

  Returns: Returns the name of the Element

- **failover**

  Description: Forces a failover from one fan to another fan

  Syntax: uint32 failover (CPowerSupply& ps_from, CPowerSupply& ps_to);

Returns: Returns The status.

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- exception

The application need to handle these exception.

Class Requirements

Header file -- powersupply.h
Library -- dashapi

3.3.14. CBattery

This class represents the Battery.

Public Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getBatteryStatus
- getBatteryStatusStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getElementName
- getChemistry
- getChemistryStr
- getMaxRechargeCount
- getRechargeCount
- getExpectedLife
- getEstimatedRunTime
- getTimeToFullCharge
- getMaxRechargeTime
- capableOfRequestStateChange

- getStatesSupported
- getStatesSupportedStr
- enable
- disable
- test
- reset

Static Member Functions

- enumBattery
- getCachedProps

Constructor Description

- **CBattery**
  Description: Constructs this object from the corresponding CIM_Battery object.
  Syntax: CBattery (const CIM_Batttery& bat);
  Parameters:
  - *bat* CIM Battery object.

Member Functions Description

- **enumBattery**
  Description: Enumerates all the batteries present under a management access point.
  Syntax: CBattery::iterator enumBattery (IClient* client,bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the battery.

- **GetCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_Battery* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**

Description: Gets the battery System Creation class Name .
Syntax: string getSystemCreationClassName (void) const;
Returns: The battery system creation class name

- **getSystemName**
  Description: Gets the battery System Name
  Syntax: string getSystemName (void) const;
  Returns: The battery system name

- **getCreationClassName**
  Description: Gets the battery Creation class Name
  Syntax: string getCreationClassName (void) const;
  Returns: The battery creation class name.

- **getDeviceID**
  Description: Gets the battery device id
  Syntax: string getDeviceID (void) const;
  Returns: The battery device id.

- **getBatteryStatus**
  Description: Gets the current status of the battery
  Syntax: uint16 getBatteryStatus (void) const;
  Returns: The current status

- **getBatteryStatusStr**
  Description: Gets the current status of the battery as string
  Syntax: string getBatteryStatusStr (void) const;
  Returns: The current status

- **getOperationalStatus**
  Description: Gets the operational status of the battery
  Syntax: vector<uint16> getOperationalStatus (void) const;
  Returns: The current status

- **getOperationalStatusStr**
  Description: Gets the operational status of the battery as string
  Syntax: vector<string> getOperationalStatusStr (void) const;
  Returns: The The operational status

- **getHealthState**
  Description: Gets the health state of the battery
  Syntax: uint16 getHealthState (void) const;
  Returns: The health state

- **getHealthStateStr**
  Description: Gets the health state of the battery as string
  Syntax: string getHealthStateStr (void) const;
  Returns: The health state

- **getEnabledState**
  Description: Gets the state of the battery
  Syntax: uint16 getEnabledState (void) const;
  Returns: The The enabled state

- **getEnabledStateStr**
  Description: Gets the state of the battery as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The enabled state

- **getRequestedState**
  Description: Gets the last requested state of the battery
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requested state

- **getRequestedStateStr**
  Description: Gets the last requested state of the battery as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The requested state

- **getElementName**
  Description: Gets the element name of the battery
  Syntax: string getElementName (void) const;
  Returns: The element name

- **getChemistry**
  Description: Gets the Chemistry.
  Syntax: uint16 getchemistry (void) const;
  Returns: The Chemistry

- **getChemistryStr**
  Description: Gets the Chemistry.
  Syntax: uint16 getchemistry (void) const;
  Returns: The battery Chemistry as string

- **getMaxRechargeCount**
  Description: Gets the MaxRechargeCount of the battery

Syntax: uint32 getMaxRechargeCount (void) const;
Returns: The MaxRechargeCount of the battery

- **getRechargeCount**
  Description: Gets the RechargeCount of the battery
  Syntax: uint32 getRechargeCount (void) const;
  Returns: The RechargeCount of the battery

- **getExpectedLife**
  Description: Gets the ExpectedLife of the battery
  Syntax: uint32 getExpectedLife (void) const;
  Returns: The ExpectedLife of the battery

- **getEstimatedRunTime**
  Description: Gets the EstimatedRunTime of the battery
  Syntax: uint32 getEstimatedRunTime (void) const;
  Returns: The EstimatedRunTime of the battery

- **getTimeToFullCharge**
  Description: Gets the TimeToFullCharge of the battery
  Syntax: uint32 getTimeToFullCharge (void) const;
  Returns: The TimeToFullCharge of the battery

- **getMaxRechargeTime**
  Description: Gets the MaxRechargeTime of the battery
  Syntax: uint32 getMaxRechargeTime (void) const;
  Returns: The MaxRechargeTime of the battery

- **capableOfRequestStateChange**
  Description: Verifies whether the Request State Change operation is supported or not.
  Syntax: ***bool capableOfRequestStateChange (void) const;***
  Returns: True or False

- **getStatesSupported**
  Description: Gets the StatesSupported values as integer vector.
  Syntax: ***vector<uint16> capableOfRequestStateChange (void) const;***
  Returns: values as integer vector.

- **getStatesSupportedStr**
- Description: Gets the StatesSupported values as string vector.
- Syntax: ***vector<string> capableOfRequestStateChange (void) const;***
- Returns: values as string vector.

- **enable**
  Description: Enable/turn on Battery
  Syntax: uint32 enable (void) const;
  Returns: None

- **disable**
  Description: Disable/turn off Battery
  Syntax: uint32 disable (void) const;
  Returns: None

- **test**
  Description: Test/perform recalculation of charge thresholds.
  Syntax: uint32 test (void) const;
  Returns: None

- **reset**
  Description: Reset/recharge of battery.
  Syntax: uint32 reset (void) const;
  Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- battery.h
Library -- dashapi

3.3.15. CBiosManagement

This class represents the Biosmanagement. It includes BIOSElements and BIOS Attributes.

Public Functions for BIOSAttribute

- getInstanceID
- getAttributeName
- getCurrentValue
- getDefaultValue

- getPendingValue
- isReadOnly
- isOrderedList
- getPossibleValues
- getPossibleValuesDescription
- getLowerBound
- getUpperBound
- getProgrammaticUnit
- getScalarIncrement
- getMaxLength
- getMinLength
- getStringType
- getStringTypeStr
- getValueExpression
- isPasswordSet
- getPasswordEncoding
- getPasswordEncodingStr
- setAttribute
- capableofBIOSManagementService
- capableOfSetBIOSAttribute
- capableOfSetBIOSAttributeEmbeddedInstance
- capableOfSetBIOSAttributes
- getSupportedEncodingsStr
- getSupportedPasswordAlgorithms

Static Member Functions for BIOSAttribute

- enumBIOSAttributes
- getCachedProps

Constructor Description for BIOSAttribute

- **CBIOSAttrubute**

  Description: Constructs this object from the corresponding CIM_BIOSAttribute object.

  Syntax: CBattery (const CIM_BIOSAttrubute& ba);

  Parameters:
  - *ba* CIM BIOSAttribute object.

Member Functions Description

- **enumBIOSAttributes**

  Description: Enumerates all the attributes present under a management access point.

Syntax: CBIOSAttribute::iterator enumBIOSAttributes (IClient* client, bool cached = true);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching. Default is true.

Returns: Iterator to the BIOSAttribute.

- **getCachedProps**

  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object
  Syntax: CIM_BIOSAttribute* getCIMObject (void);
  Returns: The underlying CIM object

- **getInstanceID**

  Description: Gets the instance id of the bios attribute
  Syntax: string getInstanceID (void) const;
  Returns: The instance id

getAttributeName

Description: Gets the AttributeName
Syntax: string getAttributeName (void) const;
Returns: The AttributeName

- **getCurrentValue**

  Description: Gets the value of the Attribute
  Syntax: vector<string> getCurrentValue (void) const;
  Returns: The current value of Attribute

- **getDefaultValue**

  Description: Gets the default value of the Attribute
  Syntax: vector<string> getDefaultValue (void)const;
  Returns: The default value of the Attribute

- **getPendingValue**

  Description: Gets an array of PendingValue
  Syntax: vector<string> getPendingValue (void) const;
  Returns: The Pending value of the Attribute

- **isReadOnly**
  Description: Gets IsReadyonly flag of the Attribte
  Syntax: boolean isReadOnly (void) const;
  Returns: true if success / false if failure

- **isOrderedList**
  Description: Gets the IsOrderedList
  Syntax: boolean isOrderedList (void) const;
  Returns: True or False

- **getPossibleValues**
  Description: Gets the possible values for this attribute.
  Syntax: vector<string> getPossibleValues (void) const;
  Returns: The possible values

- **getPossibleValuesDescription**
  Description: Gets the possible description values for this attribute
  Syntax: vector<string> getPossibleValuesDescription (void) const;
  Returns: The possible description values

- **getLowerBound**
  Description: Gets the lower bound values for this attribute
  Syntax: uint64 getLowerBound (void) const;
  Returns: The lower bound values

- **getUpperBound**
  Description: Gets the upper bound values for this attribute
  Syntax: uint64 getUpperBound (void) const;
  Returns: The upper bound values

- **getProgrammaticUnit**
  Description: Gets the programmatic unit for this attribute
  Syntax: string getProgrammaticUnit (void) const;
  Returns: The The programmatic unit

- **getScalarIncrement**
  Description: Gets the scalar increment for this attribute
  Syntax: uint32 getScalarIncrement (void) const;
  Returns: The scalar increment

- **getMaxLength**
  Description: Gets the maximum length of string for this attribute

Syntax: uint64 getMaxLength (void) const;
Returns: The maximum length

- **getMinLength**
  Description: Gets the minimum length of string for this attribute
  Syntax: uint64 getMinLength (void) const;
  Returns: The minimum length

- **getStringType**
  Description: Gets the string type for this attribute.
  Syntax: uint32 getStringType (void) const;
  Returns: The string type

- **getStringTypeStr**
  Description: Gets the string type for this attribute as string
  Syntax: string getStringTypeStr (void) const;
  Returns: The string type

- **getValueExpression**
  Description: Gets the ValueExpression of string for this attribute
  Syntax: string getValueExpression (void) const;
  Returns: The ValueExpression

- **isPasswordSet**
  Description: Checks if password is set for this attribute
  Syntax: boolean isPasswordSet (void) const;
  Returns: The true or false

- **getPasswordEncoding**
  Description: Gets the PasswordEncoding type for this attribute.
  Syntax: uint32 getPasswordEncoding (void) const;
  Returns: The PasswordEncoding

- **getPasswordEncodingStr**
  Description: Gets the PasswordEncoding for this attribute as string
  Syntax: string getPasswordEncodingStr (void) const;
  Returns: The PasswordEncoding

- **setAttribute**
  Description: Sets the Bios Attribute
  Syntax: uint32 setAttribute (const vector <string>& value);

- **capableofBIOSManagementService**
  Description: Verifies whether BIOS Management Service and Capabilities instances are exists or not.
  Syntax: ***bool capableOfBIOSManagementService (void) const;***
  Returns: True or False.

- **capableofBIOSManagementService**
  Description: Verifies whether BIOS Management Service and Capabilities instances are exists or not.
  Syntax: ***bool capableOfBIOSManagementService (CIM_BIOSService &bs, CIM_BIOSServiceCapabilities &bsc) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.
  - *CIM_BIOSServiceCapabilities* Output parameter passed as reference.

  Returns: True or False.

- **capableOfSetBIOSAttribute**
  Description: Verifies whether the SetBIOSAttribute operation is supported or not.
  Syntax: ***bool capableOfSetBIOSAttribute (void) const;***
  Returns: True or False.

- **capableOfSetBIOSAttribute**
  Description: Verifies whether the SetBIOSAttribute operation is supported or not.
  Syntax: ***bool capableOfSetBIOSAttribute (CIM_BIOSService &bs, CIM_BIOSElement &be) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.
  - *CIM_BIOSElement* Output parameter passed as reference.

  Returns: True or False.

- **capableOfSetBIOSAttributeEmbeddedInstance**
  Description: Verifies whether the SetBIOSAttributeEmbeddedInstance operation is supported or not.
  Syntax: ***bool capableOfSetBIOSAttributeEmbeddedInstance (void) const;***
  Returns: True or False.

- **capableOfSetBIOSAttributeEmbeddedInstance**
  Description: Verifies whether the SetBIOSAttribute EmbeddedInstance operation is supported or not.
  Syntax: ***bool capableOfSetBIOSAttributeEmbeddedInstance (CIM_BIOSService***

&bs, CIM_BIOSElement &be) const;

Parameters:

- *CIM_BIOSService* Output parameter passed as reference.
- *CIM_BIOSElement* Output parameter passed as reference.

Returns: True or False.

- **capableOfSetBIOSAttributes**

  Description: Verifies whether the SetBIOSAttributes operation is supported or not.

  Syntax: ***bool capableOfSetBIOSAttributes (void) const;***

  Returns: True or False.

- **capableOfSetBIOSAttributes**

  Description: Verifies whether the SetBIOSAttributes operation is supported or not.

  Syntax: ***bool capableOfSetBIOSAttributes (CIM_BIOSService &bs, CIM_BIOSElement &be) const;***

  Parameters:

  - *CIM_BIOSService* Output parameter passed as reference.
  - *CIM_BIOSElement* Output parameter passed as reference.

  Returns: True or False.

- **getSupportedEncodingsStr**

  Description: Gets the Supported Password Encodings as string values.

  Syntax: ***bool getSupportedEncodingsStr (vector<string> &str) const;***

  Parameters:

  - *vector<string>* Output parameter passed as reference.

  Returns: True or False.

- **getSupportedPasswordAlgorithms**

  Description: Gets the Supported Password Algorithms as string values.

  Syntax: ***bool getSupportedPasswordAlgorithms (vector<string> &str) const;***

  Parameters:

  - *vector<string>* Output parameter passed as reference.

  Returns: True or False.

Public Functions for BIOSElement

- getManufacturer
- getPrimaryBIOS
- getVersion
- getName

- getSoftwareElementState
- getSoftwareElementStateStr
- getSoftwareElementID
- getTargetOperatingSystem
- getTargetOperatingSystemStr
- getRegistryURIs
- getAttributes
- restoreDefaults
- capableofBIOSManagementService
- isSupportedMethod
- capableOfRestoreDefault
- capableOfReadRawBIOSData
- capableOfWriteRawBIOSData
- getsSupportedMethods

Static Member Functions for BIOSElement

- enumBIOSElements
- getCachedProps

Constructor Description for BIOSElement

- **CBIOSElement**
  Description: Constructs this object from the corresponding CIM_BIOSElement object.
  Syntax: CBattery (const CIM_BIOSElement& be);
  Parameters:
  - *be* CIM BIOSElement object.

Member Functions Description

- **enumBIOSElements**
  Description: Enumerates all the Elements present under a management access point.
  Syntax: CBIOSElement::iterator enumBIOSElements (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the BIOSAttribute.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);

Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_BIOSElement* getCIMObject (void);
  Returns: The underlying CIM object

- **getManufacturer**
  Description: Gets the Manufacturer of this BIOS
  Syntax: string getManufacturer (void) const;
  Returns: The Manufacturer of this BIOS

- **getPrimaryBIOS**
  Description: Gets the PrimaryBIOS
  Syntax: boolean getPrimaryBIOS (void) const;
  Returns: return True False

- **getVersion**
  Description: Gets the version of the BIOS
  Syntax: string getVersion (void) const;
  Returns: The Version

- **getName**
  Description: Gets the name of the Attribute
  Syntax: string getName (void) const;
  Returns: The name of the bios attribute

- **getSoftwareElementState**
  Description: Gets the SoftwareElementState
  Syntax: uint16 getSoftwareElementState (void) const;
  Returns: The SoftwareElementState

- **getSoftwareElementStateStr**
  Description: Gets the SoftwareElementState as string
  Syntax: string getSoftwareElementStateStr (void) const;
  Returns: The SoftwareElementState as string

- **getSoftwareElementID**
  Description: Gets the SoftwareElementID
  Syntax: string getSoftwareElementID (void) const;
  Returns: The SoftwareElementID

- **getTargetOperatingSystem**

Description: Gets the TargetOperatingSystem
Syntax: uint16 getTargetOperatingSystem (void) const;
Returns: The TargetOperatingSystem

- **getTargetOperatingSystemStr**
  Description: Gets the TargetOperatingSystem as string
  Syntax: string getTargetOperatingSystemStr (void) const;
  Returns: The TargetOperatingSystem

- **getRegistryURIs**
  Description: Gets the Registry URI's
  Syntax: vector<string> getRegistryURIs (void) const;
  Returns: The Registry URI's

- **getAttributes**
  Description: Gets the Bios Attributes
  Syntax: vector<CBIOSAttribute> getAttributes (void) const;
  Returns: The Bios attributes

- **restoreDefaults**
  Description: Restore the BIOSDefaults values
  Syntax: uint32 restoreDefaults (void);
  Returns: None

- **capableofBIOSManagementService**
  Description: Verifies whether BIOS Management Service and Capabilities instances are exists or not.
  Syntax: ***bool capableOfBIOSManagementService (CIM_BIOSService &bs, CIM_BIOSServiceCapabilities &bsc) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.
  - *CIM_BIOSServiceCapabilities* Output parameter passed as reference.

  Returns: True or False.

- **isSupportedMethod**
  Description: Verifies whether particular Method is supported or not.
  Syntax: ***bool capableOfBIOSManagementService (CIM_BIOSService &bs, uint32 val) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.
  - *Uint32* Integer value of a Method.

  Returns: True or False.

- **capableOfRestoreDefault**
  Description: Verifies whether the RestoreDefault operation is supported or not.
  Syntax: ***bool capableOfRestoreDefault (void) const;***
  Returns: True or False.

- **capableOfRestoreDefault**
  Description: Verifies whether the RestoreDefault operation is supported or not.
  Syntax: ***bool capableOfRestoreDefault (CIM_BIOSService &) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.

  Returns: True or False.

- **capableOfReadRawBIOSData**
  Description: Verifies whether the ReadRawBIOSData operation is supported or not.
  Syntax: ***bool capableOfReadRawBIOSData(void) const;***
  Returns: True or False.

- **capableOfReadRawBIOSData**
  Description: Verifies whether the ReadRawBIOSData operation is supported or not.
  Syntax: ***bool capableOfReadRawBIOSData (CIM_BIOSService &) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.

  Returns: True or False.

- **capableOfWtiteRawBIOSData**
  Description: Verifies whether the WriteRawBIOSData operation is supported or not.
  Syntax: ***bool capableOfWriteRawBIOSData(void) const;***
  Returns: True or False.

- **capableOfWriteRawBIOSData**
  Description: Verifies whether the WriteRawBIOSData operation is supported or not.
  Syntax: ***bool capableOfWriteRawBIOSData (CIM_BIOSService &) const;***
  Parameters:
  - *CIM_BIOSService* Output parameter passed as reference.

  Returns: True or False.

- **getSupportedMethods**

Description:	Gets the the Integer values of Supported Methods.
Syntax:	***vector<uint32> getSupportedMethods(void) const;***
Returns:	Integer vector of Supported Methods.

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionNotSupported
- EfunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- biosmanagement.h
Library -- dashapi

3.3.16. CDHCPClient

This class represents the DHCPClient.

Public Functions

- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getNameFormat
- getProtocolIFType
- getProtocolIFTypeStr
- getOtherTypeDescription
- getRequestedState
- getRequestedStateStr
- getEnabledState
- getEnabledStateStr
- getClientState
- getElementName

Static Member Functions

- enumDHCPClient
- getCachedProps

Constructor Description

- **CDHCPClient**
  Description: Constructs this object from the corresponding CIM_DHCPProtocolEndpoint object.
  Syntax: CDHCPClient (const CIM_DHCPProtocolEndpoint& dhcp);
  Parameters:
  - *dhcp* CIM DHCPProtocolEndPoint object.

Member Functions Description

- **enumDHCPClient**
  Description: Enumerates all the DHCPClients present under a management access point.
  Syntax: CDHCPClient::iterator enumDHCPClient (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the DHCPClient.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_DHCPProtocolEndpoint* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**
  Description: Gets the SystemCreationClassName
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The systemcreationclassname

- **getCreationClassName**
  Description: Gets the CreationClassName.
  Syntax: string getCreationClassName (void) const;
  Returns: The CreationClassName

- **getSystemName**
  Description: Gets the SystemName
  Syntax: string getSystemName (void) const;
  Returns: The SystemName

- **getName**
  Description: Gets the dhcpclient name
  Syntax: string getName (void) const;
  Returns: The dhcpclient name

- **getNameFormat**
  Description: Gets the NameFormat.
  Syntax: string getNameFormat (void) const;
  Returns: The NameFormat

- **getProtocolIFType**
  Description: Gets the ProtocolIFType.
  Syntax: uint16 getProtocolIFType (void) const;
  Returns: The ProtocolIFType

- **getProtocolIFTypeStr**
  Description: Gets the ProtocolIFType as string.
  Syntax: string getProtocolIFTypeStr (void) const;
  Returns: The ProtocolIFType
- **getOtherTypeDescription**
  Description: Gets Protocol IP type description if the ProtocolIPType contains "Other"
  Syntax: string getOtherTypeDescription (void) const;
  Returns: The protocol type description.

- **getRequestedState**
  Description: Gets the RequestedState.
  Syntax: uint16 getRequestedState (void) const;
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the RequestedState as string.
  Syntax: string getRequestedStateStr (void) const;
  Returns: The RequestedState as string

- **getEnabledState**
  Description: Gets the EnabledState.
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState

Syntax: string getEnabledStateStr (void) const;
Returns: The Enabled State as string

- **getClientState**
  Description: Gets the ClientState
  Syntax: string getClientState (void) const;
  Returns: The ClientState

- **getElementName**
  Description: Gets the ElementName
  Syntax: string getElementName (void) const;
  Returns: The ElementName

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- dhcpclient.h
Library -- dashapi

3.3.17 CDNSClient

This class represents DNSClient.

Public Functions

- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getNameFormat
- getHostname
- getProtocolIFType
- getProtocolIFTypeStr
- getRequestedState
- getRequestedStateStr
- getEnabledState
- getEnabledStateStr
- getElementName

- appendPrimarySuffixes
- appendParentSuffixes
- getDNSSuffixesToAppend
- getDomainName
- useSuffixWhenRegistering
- registerThisConnectionsAddress
- getDHCPOptionsToUse

Static Member Functions

- enumDNSClient
- getCachedProps

Constructor Description

- **CDNSClient**
  Description: **Constructs this object from the corresponding CIM_DNSProtocolEndpoint object.**
  Syntax: **CDNSClient (const CIM_DNSProtocolEndpoint& dns);**
  Parameters:
  - *dns* **CIM DNSProtocolEndPoint object.**

Member Functions Description

- **enumDNSClient**
  Description: Enumerates all the DNSClients present under a management access point.
  Syntax: CDNSClient::iterator enumDNSClient (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the DNSClient.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_DNSProtocolEndpoint* getCIMObject (void);
  Returns: The underlying CIM object
- **getSystemCreationClassName**

Description: Gets the System Creation class of the dnsclient
Syntax: string getSystemCreationClassName (void);
Returns: The System Creation Class name

- **getCreationClassName**
  Description: Gets the CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The Creation class name

- **getSystemName**
  Description: Gets the System name of the dnsclient
  Syntax: string getSystemName (void);
  Returns: The System name

- **getName**
  Description: Gets the Name.
  Syntax: string getName (void);
  Returns: The Name

- **getNameFormat**
  Description: Gets the NameFormat.
  Syntax: string getNameFormat (void);
  Returns: The NameFormat

- **getHostName**
  Description: Gets the HostName.
  Syntax: string getHostName(void);
  Returns: The HostName

- **getProtocolIFType**
  Description: Gets the ProtocolIfType.
  Syntax: uint16 getProtocolIFType(void);
  Returns: The ProtocolIfType

- **getProtocolIFTypeStr**
  Description: Gets the getProtocolIFType as string.
  Syntax: string getProtocolIFTypeStr (void);
  Returns: The ProtocolIFType string

- **getRequestedState**
  Description: Gets the RequestedState
  Syntax: uint16 getRequestedState (void);

Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the RequestedState as string
  Syntax: string getRequestedStateStr (void);
  Returns: The RequestedState string

- **getEnabledState**
  Description: Gets the EnabledState
  Syntax: uint16 getEnabledState (void);
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState as string
  Syntax: string getEnabledStateStr (void);
  Returns: The EnabledState string

- **getElementName**
  Description: Gets the ElementName
  Syntax: string getElementName (void);
  Returns: The ElementName

- **getAppendPrimarySuffixes**
  Description: Gets AppendPrimarySuffixes
  Syntax: boolean getAppendPrimarySuffixes (void) const;
  Returns: true if success
  false otherwise

- **getAppendParentSuffixes**
  Description: Gets AppendParentSuffixes
  Syntax: boolean getAppendParentSuffixes (void) const;
  Returns: true if success
  false otherwise

- **getDNSSuffixesToAppend**
  Description: Gets an array of DNSSuffixesToAppend.
  Syntax: vector<string> getDNSSuffixesToAppend (void);
  Returns: list of DNSSuffixesToAppend

- **getDomainName**
  Description: Gets the Domain Name.
  Syntax: string getDomainName (void);
  Returns: The DomainName

- **useSuffixWhenRegistering**
  Description: Gets the UseSuffixWhenRegistering
  Syntax: boolean useSuffixWhenRegistering (void);
  Returns: True/False

- **registerThisConnectionsAddress**
  Description: Gets the registerThisConnectionsAddress.
  Syntax: boolean registerThisConnectionsAddress (void);
  Returns: True/False

- **getDHCPOptionsToUse**
  Description: Gets the an array of DHCPOptionsToUse.
  Syntax: vector<uint16> getDHCPOptionsToUse (void);
  Returns: list of DHCPOptionsToUse.

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- dnsclient.h
Library -- dashapi

3.3.18. CIPInterface

This class represents IPInterface.

Public Functions

- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getNameFormat
- getProtocolIFType
- getProtocolIFTypeStr
- getRequestedState
- getRequestedStateStr
- getEnabledState
- getEnabledStateStr

- getElementName
- getIPv4Address
- getSubnetMask
- getAddressOrigin
- getAddressOriginStr
- getIPv6Address
- getIPv6AddressType
- getIPv6AddressTypeStr
- getIPv6SubnetPrefixLength

Static Member Functions

- enumIPInterface
- getCachedProps

Constructor Description

- **CIPInterface**
  Description: Constructs this object from the corresponding CIM_IPProtocolEndpoint object.
  Syntax: CDHCPClient (const CIM_IPProtocolEndpoint& IP);
  Parameters:
  - *IP* CIM IPProtocolEndPoint object.

Member Functions Description

- **enumIPInterface**
  Description: Enumerates all the IPInterfaces present under a management access point.
  Syntax: CIPInterface::iterator enumIPInterface (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the IPInterface.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object

Syntax: CIM_IPProtocolEndpoint* getCIMObject (void);
Returns: The underlying CIM object

- **getSystemCreationClassName**
  Description: Gets the System Creation class of the ipinterface
  Syntax: string getSystemCreationClassName (void);
  Returns: The System Creation Class name
- **getCreationClassName**
  Description: Gets the CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The Creation class name

- **getSystemName**
  Description: Gets the System name of the ipinterface
  Syntax: string getSystemName (void);
  Returns: The System name

- **getName**
  Description: Gets the Name.
  Syntax: string getName (void);
  Returns: The Name

- **getNameFormat**
  Description: Gets the NameFormat.
  Syntax: string getNameFormat (void);
  Returns: The NameFormat

- **getProtocolIFType**
  Description: Gets the ProtocolIfType.
  Syntax: uint16 getProtocolIFType(void);
  Returns: The ProtocolIfType

- **getProtocolIFTypeStr**
  Description: Gets the getProtocolIFType as string.
  Syntax: string getProtocolIFTypeStr (void);
  Returns: The ProtocolIFType string
- **getRequestedState**
  Description: Gets the RequestedState
  Syntax: uint16 getRequestedState (void);
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the RequestedState as string

Syntax: string getRequestedStateStr (void);
Returns: The RequestedState string

- **getEnabledState**
  Description: Gets the EnabledState
  Syntax: uint16 getEnabledState (void);
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState as string
  Syntax: string getEnabledStateStr (void);
  Returns: The EnabledState string

- **getElementName**
  Description: Gets the ElementName
  Syntax: string getElementName (void);
  Returns: The ElementName

- **getIPv4Address**
  Description: Gets the getIPv4Address.
  Syntax: string getIPv4Address (void);
  Returns: The IPv4Address

- **getSubnetMask**
  Description: Gets the getSubnetMask.
  Syntax: string getSubnetMask (void);
  Returns: The SubnetMask

- **getAddressOrigin**
  Description: Gets the getAddressOrigin.
  Syntax: uint16 getAddressOrigin (void);
  Returns: The AddressOrigin

- **getAddressOriginStr**
  Description: Gets the AddressOrigin as string.
  Syntax: string getAddressOriginStr (void);
  Returns: The AddressOrigin string

- **getIPv6Address**
  Description: Gets the getIPv6Address.
  Syntax: string getIPv6Address (void);
  Returns: The IPv6Address

- **getIPv6AddressType**
  Description: Gets the getIPv6AddressType.
  Syntax: uint16 getIPv6AddressType (void);
  Returns: The IPv6AddressType

- **getIPv6AddressTypeStr**
  Description: Gets the getIPv6AddressType as string.
  Syntax: string getIPv6AddressTypeStr (void);
  Returns: The IPv6AddressType string

- **getIPv6SubnetPrefixLength**
  Description: Gets the getIPv6SubnetPrefixLength.
  Syntax: uint16 getIPv6SubnetPrefixLength (void);
  Returns: The IPv6SubnetPrefixLength

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- ipinterface.h
Library -- dashapi

3.3.19. CNetworkPort

This class represents NetworkPort.

Public Functions

- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getSpeed
- getLinkTechnology
- getLinkTechnologyStr
- getPermanentAddress
- getMaxSpeed
- getRequestedSpeed
- getDeviceID

- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getElementName

Static Member Functions

- enumNetworkPorts
- getCachedProps

Constructor Description

- **CNetworkPort**
  Description: Constructs this object from the corresponding CIM_NetworkPort object.
  Syntax: CNetworkPort (const CIM_NetworkPort& np);
  Parameters:
  - *np* CIM NetworkPort object.

Member Functions Description

- **enumNetworkPorts**
  Description: Enumerates all the NetworkPorts present under a management access point.
  Syntax: CNetworkPort::iterator enumNetworkPorts (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the NetworkPort.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_NetworkPort* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**
  Description: Gets the System Creation class of the networkport

Syntax: string getSystemCreationClassName (void);
Returns: The System Creation Class name

- **getCreationClassName**
  Description: Gets the CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The Creation class name

- **getSystemName**
  Description: Gets the System name of the networkport
  Syntax: string getSystemName (void);
  Returns: The System name

- **getName**
  Description: Gets the Name.
  Syntax: string getName (void);
  Returns: The Name

- **getSpeed**
  Description: Gets the Speed.
  Syntax: uint64 getSpeed (void);
  Returns: The speed

- **getLinkTechnology**
  Description: Gets the LinkTechnology.
  Syntax: uint16 getLinkTechnology (void);
  Returns: The LinkTechnology

- **getLinkTechnologyStr**
  Description: Gets the LinkTechnology as string.
  Syntax: string getLinkTechnologyStr (void);
  Returns: The LinkTechnology string

- **getPermanentAddress**
  Description: Gets the PermanentAddress.
  Syntax: string getPermanentAddress (void);
  Returns: The PermanentAddress

- **getMaxSpeed**
  Description: Gets the MaxSpeed.
  Syntax: uint64 getMaxSpeed (void);
  Returns: The MaxSpeed

- **getRequestedSpeed**
  Description: Gets the RequestedSpeed.
  Syntax: uint64 getRequestedSpeed (void);
  Returns: The RequestedSpeed

- **getDeviceID**
  Description: Gets the Device ID.
  Syntax: string getDeviceID (void);
  Returns: The Device ID

- **getRequestedState**
  Description: Gets the RequestedState
  Syntax: uint16 getRequestedState (void);
  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the RequestedState as string
  Syntax: string getRequestedStateStr (void);
  Returns: The RequestedState string

- **getEnabledState**
  Description: Gets the EnabledState
  Syntax: uint16 getEnabledState (void);
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState as string
  Syntax: string getEnabledStateStr (void);
  Returns: The EnabledState string

- **getElementName**
  Description: Gets the ElementName
  Syntax: string getElementName (void);
  Returns: The ElementName

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- ipinterface.h
Library -- dashapi

3.3.20. COpaqueManagementData

This a class that represents OpaqueManagementData.

Public Functions

- getTransformedDataSize
- getMaxSize
- getUntransformedDataFormat
- getTransformations
- getTransformationKeyIDs
- getLastAccessed
- getWriteLimited
- getDataOrganization
- getAccess
- getNumberOfBlocks
- getConsumableBlocks
- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- readData
- writeData

Static Member Functions

- enumOpaqueManagementData
- getCachedProps

Constructor Description

- **COpaqueManagementData**
  Description: Constructs this object from the corresponding CIM_OpaqueManagementData object.
  Syntax: COpaqueManagementData (constCIM_OpaquwManagementData&omd);
  Parameters:
  - *omd* CIM OpaqueManagementData object.

Member Functions Description

- **enumOpaqueManagementData**
  Description: Enumerates all the Opaque Management Data present under a management access point.
  Syntax: COpaqueManagementData::iterator enumOpaqueManagementSata (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the OpaqueManagementData.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_OpaqueManagementData* getCIMObject (void);
  Returns: The underlying CIM object

- **getTransformedDataSize**
  Description: Gets the Transformed Data Size.
  Syntax: uint64 getTransformedDataSize (void);
  Returns: The Transformed data size

- **getMaxSize**
  Description: Gets Maximum Size.
  Syntax: uint64 getMaxSize (void);
  Returns: The MaxSize

- **getUntransformedDataFormat**
  Description: Gets the Untransformed Data Format.
  Syntax: uint16 getUntransformedDataFormat (void);
  Returns: The Untransformed Data Format

- **getTransformations**
  Description: Gets the an array of Transformations
  Syntax: vector<uint16> getTransformations (void);
  Returns: The list of of Transformations

- **getTransformationKeyIDs**
  Description: Gets the array of TransformationKeyIDs.
  Syntax: vector<string> getTransformationKeyIDs (void);
  Returns: The list of of TransformationKeyIDs

- **getLastAccessed**
  Description: Gets the LastAccessed data
  Syntax: datetime getLastAccessed (void);
  Returns: The LastAccessed data

- **getWriteLimited**
  Description: Gets the WriteLimited value.
  Syntax: uint16 getWriteLimited (void);
  Returns: The WriteLimited

- **getDataOrganization**
  Description: Gets the DataOrganization value.
  Syntax: uint16 getDataOrganization(void);
  Returns: The DataOrganization value

- **getAccess**
  Description: Gets the Access data
  Syntax: uint16 getAccess (void);
  Returns: The Access data

- **getNumberOfBlocks**
  Description: Gets the Number of Blocks.
  Syntax: uint64 getNumberofblocks (void);
  Returns: The Number of Blocks

- **getBlockSize**
  Description: Gets the BlockSize.
  Syntax: uint64 getBlockSize (void);
  Returns: The BlockSize

- **getConsumableBlocks**
  Description: Gets the ConsumableBlocks.
  Syntax: uint64 getConsumableBlocks (void);
  Returns: The ConsumableBlocks

- **getSystemCreationClassName**
  Description: Gets the System Creation class of the opaquemanagementdata
  Syntax: string getSystemCreationClassName (void);

Returns: The System Creation Class name

- **getSystemName**
  Description: Gets the System name of the opaquemanagementdata
  Syntax: string getSystemName (void);
  Returns: The System name

- **getCreationClassName**
  Description: Gets the CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The Creation class name

- **getDeviceID**
  Description: Gets the Device ID.
  Syntax: string getDeviceID (void);
  Returns: The Device ID

- **readData**
  Description: Reads the Data.
  Syntax: uint32 readData (uint64 offset, uint64* length,
  vector<uint8>* data);
  Parameters:
  - *offset* Offset of the memory location.
  - *length* length of the data.
  - *data* input given by the user.

  Returns: The Data

- **writeData**
  Description: Writes the Data.
  Syntax: uint32 writeData (uint64 offset, uint64* length,
  vector<uint8>* data);
  Parameters:
  - *offset* Offset of the memory location.
  - *length* length of the data.
  - *data* input given by the user.

  Returns: The Data

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EfunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on

exceptions

Class Requirements

Header file -- opaquemanagementdata.h
Library -- dashapi

3.3.21. COperatingSystem

This class represents OperatingSystem

Public Functions

- getCSCreationClassName
- getCSName
- getCreationClassName
- getName
- getOSType
- getOtherTypeDescription
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getAvailableRequestedStates
- getAvailableRequestedStatesStr
- getTransitioningToState
- getTransitioningToStateStr

Static Member Functions

- enumOperatingSystems
- getCachedProps

Constructor Description

- **COperatingSystem**
  Description: Constructs this object from the corresponding CIM_OperatingSystem object.
  Syntax: COperatingSystem (const CIM_OperatingSystem& os);
  Parameters:
  - *os* CIM OperatingSystem object.

Member Functions Description

- **senumOperatingSystems**

  Description: Enumerates all the OperatingSystems present under a management access point.

  Syntax: COperatingSystem::iterator enumOperatingSystems (IClient* client, bool cached = true);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the OperatingSystem.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_OperatingSystem* getCIMObject (void);

  Returns: The underlying CIM object

- **getCSCreationClassName**

  Description: Gets the Operating System CSCreationClassName.

  Syntax: string getCSCreationClassName (void) const;

  Returns: The Operating System CSCreationClassName

- **getCSName**

  Description: Gets the Operating System CSName

  Syntax: string getCSName (void) const;

  Returns: The Operating System CSName.

- **getCreationClassName**

  Description: Gets the Operating System CreationClassName

  Syntax: string getCreationClassName (void) const;

  Returns: The Operating System CreationClassName

- **getName**

  Description: Gets the operationg system Name.

  Syntax: string getName (void) const;

  Returns: The operating system Name

- **getName**

Description: Gets the operationg system Name.
Syntax: string getName (void) const;
Returns: The operating system Name

- **getOSType**
  Description: Gets the operating system type.
  Syntax: string getOSType (void) const;
  Returns: The operating system type

- **getOtherTypeDescription**
  Description: Gets the OS type description if the OSType contains "Other".
  Syntax: string getOtherTypeDescription (void) const;
  Returns: The Other OSType description

- **getEnabledState**
  Description: Gets the EnabledState
  Syntax: uint16 getEnabledState (void);
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the EnabledState as string
  Syntax: string getEnabledStateStr (void);
  Returns: The EnabledState string

- **getRequestedState**
  Description: Gets the RequestedState
  Syntax: uint16 getRequestedState (void);
  Returns: The RequestedState
- **getRequestedStateStr**
  Description: Gets the RequestedState as string
  Syntax: string getRequestedStateStr (void);
  Returns: The RequestedState string

- **getAvailableRequestedStates**
  Description: Gets the available requested states of the operating system.
  Syntax: vector<uint16> getAvailableRequestedStates (void) const;
  Returns: The available requested states

- **getAvailableRequestedStatesStr**
  Description: Gets the available requested states of the operating system as string.
  Syntax: vector<string> getAvailableRequestedStatesStr (void) const;

Returns: The available requested states

- **getTransitioningToState**
  Description: Gets the transitioning state of operating system.
  Syntax: uint16 getTransitioningToState (void) const;
  Returns: The transistioning state

- **getTransitioningToState**
  Description: Gets the transitioning state of operating system.
  Syntax: uint16 getTransitioningToState (void) const;
  Returns: The transistioning state

- **getTransitioningToStateStr**
  Description: Gets the transitioning state of operating system as string.
  Syntax: string getTransitioningToStateStr (void) const;
  Returns: The transistioning state

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- operatingsystem.h
Library -- dashapi

3.3.22. CTextRedirection

This class represents TextRedirection.

Public Functions

- getSystemCreationClassName
- getSystemName
- getName
- getCreationClassName
- getElementName
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getTerminationSequence

- getTextFlowType
- getPortNumber
- getProtocolIFType
- capableOfTRServiceREquestStateChange
- capableOfTRSAPRequestStateChange
- isSupportedRequestedState
- getSupportedStates
- getSupportedStatesStr
- activate
- enable
- disable
- startRedirection

Static Member Functions

- enumTextRedirections
- getCachedProps

Constructor Description

- **CTextRedirecton**
  Description: Constructs this object from the corresponding CIM_TextRedirectionSAP object.
  Syntax: CTextRedirection (const CIM_TextRedirectionSAP& tr);
  Parameters:
  - *tr* CIM TextRedirectionSAP object.

Member Functions Description

- **enumTextRedirections**
  Description: Enumerates all the TextRedirectons present under a management access point.
  Syntax: CTextRedirection::iterator enumTextRedirections (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the TextRedirection.

- **getCachedProps**
  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_TextRedirectionSAP* getCIMObject (void);
  Returns: The underlying CIM object
- **getSystemCreationClassName**
  Description: Gets System Creation ClassName
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The systemcreationclassname

- **getSystemName**
  Description: Gets SystemName
  Syntax: string getSystemName (void) const;
  Returns: The system name

- **getName**
  Description: Gets the Text redirection name
  Syntax: string getName (void) const;
  Returns: The textredirection name

- **getCreationClassName**
  Description: Gets CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The creationclassname

- **getElementName**
  Description: Gets ElementName
  Syntax: string getElementName (void) const;
  Returns: The ElementName

- **getEnabledState**
  Description: Gets EnabledState
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets EnabledState as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The Enabled State

- **getRequestedState**
  Description: Gets RequestedState
  Syntax: uint16 getRequestedState (void) const;

Returns: The Requested State

- **getRequestedStateStr**
  Description: Gets Requested State as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The Requested State as string

- **getTerminationSequence**
  Description: Gets the session terminate sequence
  Syntax: string getTerminationSequence (void) const;
  Returns: The termination sequence

- **getTextFlowType**
  Description: Gets the text flow type
  Syntax: string getTextFlowType (void) const;
  Returns: The text flow type

- **getPortNumber**
  Description: Gets the PortNumber
  Syntax: uint32 getPortNumber (void) ;
  Returns: PortNumber

- **getProtocolIFType**
  Description: Gets the ProtocolIFType
  Syntax: string getProtocolIFType (void) ;
  Returns: ProtocolIFType

- **capableOfTRServiceRequestStateChange**
  Description: Verifies whether the TRService RequestStateChange operation is supported or not.
  Syntax: ***bool capableOfTRServiceRequestedStateChange (void) ;***
  Returns: True or False

- **capableOfTRServiceRequestStateChange**
  Description: Verifies whether the TRService RequestStateChange operation is supported or not.
  Syntax: *bool* ***capableOfTRServiceRequestedStateChange (CIM_TextRedirectionService &te) ;***
  Parameters:
    - *CIM_TextRedirectionService* Output parameter passed as reference.

  Returns: True or False

- **capableOfTRSAPRequestStateChange**

Description: Verifies whether the TRSAP RequestStateChange operation is supported or not.
Syntax: ***bool capableOfTRSAPRequestedStateChange (void) ;***
Returns: True or False

- **capableOfTRSAPRequestStateChange**

Description: Verifies whether the TRSAP RequestStateChange operation is supported or not.
Syntax: *bool* ***capableOfTRSAPRequestedStateChange (CIM_TextRedirectionService &te) ;***
Parameters:
- *CIM_TextRedirectionService* Output parameter passed as reference.

Returns: True or False

- **isSupportedRequestedState**

Description: Verifies whether the perticular Requested States property is supported or not.
Syntax: *bool* ***isSupportedReuestedState (CIM_TextRedirectionService &te, uint16 val) const ;***
Parameters:
- *CIM_TextRedirectionService* Output parameter passed as reference.
- *Uint16* Integer value of RequestedState.

Returns: True or False

- **getSupportedStates**

Description: Gets the SupportedStates values as Integer vector.
Syntax: *bool****igetSupportedStates (vector<uint16> &val) const ;***
Parameters:
- *Uint16* Integer vectore passed as reference to return value of SupportedStates.

Returns: True or False

- **getSupportedStatesStr**

Description: Gets the SupportedStates values as String vector.
Syntax: *bool****igetSupportedStatesStr (vector<string> &val) const ;***
Parameters:
- *Uint16* String vectore passed as reference to return value of SupportedStates.

Returns: True or False

- **activate**

Description: Activates the Redirection.

Syntax: void activate (void) const;
Returns: None

- **enable**
  Description: Service is enabled but not active.
  Syntax: void enable (void) const;
  Returns: None

- **disable**
  Description: Disable this redirection session
  Syntax: void disable (void) const;
  Returns: None

- **StartRedirection**
  Description: Starts the Redirection.
  Syntax: int StartRedirection (void);
  Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionNotSupported
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- textredirection.h
Library -- dashapi

3.3.23. CUSBRedirection

This class represents USBRedirection.

Public Functions

- getSystemCreationClassName
- getSystemName
- getName
- getCreationClassName
- getElementName
- getEnabledState

- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getConnectionMode
- getResetTimeout
- getSessionTimeout
- capableOfUSBRServiceREquestStateChange
- capableOfUSBRSAPRequestStateChange
- isSupportedRequestedState
- getSupportedStates
- getSupportedStatesStr
- activate
- enable
- disable
- startFolderRedirection

Static Member Functions

- enumUSBRedirections
- getCachedProps

Constructor Description

- **CUSBRedirections**
  Description: Constructs this object from the corresponding CIM_OperatingSystemSAP object.
  Syntax: CUSBRedirection (const CIM_USBRedirectionSAP& usbr);
  Parameters:
  - *usbr* CIM USBRedirectionSAP object.

Member Functions Description

- **enumUSBRedirections**
  Description: Enumerates all the USBRedirections present under a management access point.
  Syntax: CUSBRedirection::iterator enumUSBRedirections (IClient* client, bool cached = true);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the USBRedirection.

- **getCachedProps**
  Description: Gets the properties that are cached by this object

Syntax: vector< string > getCachedProps (void);
Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_USBRedirectionSAP* getCIMObject (void);
  Returns: The underlying CIM object

- **getSystemCreationClassName**
  Description: Gets SystemCreationClassName
  Syntax: string getSystemCreationClassName (void) const;
  Returns: The systemcreationclassname

- **getSystemName**
  Description: Gets SystemName
  Syntax: string getSystemName (void) const;
  Returns: The systemname

- **getName**
  Description: Gets the usb redirection name
  Syntax: string getName (void) const;
  Returns: The usbredirection name

- **getCreationClassName**
  Description: Gets CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The creationclassname

- **getElementName**
  Description: Gets ElementName
  Syntax: string getElementName (void) const;
  Returns: None

- **getEnabledState**
  Description: Gets EnabledState
  Syntax: uint16 getEnabledState (void) const;
  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets EnabledState as string
  Syntax: string getEnabledStateStr (void) const;
  Returns: The EnabledState

- **getRequestedState**
  Description: Gets RequestedState
  Syntax: uint16 getRequestedState (void) const;
  Returns: The requestedstate

- **getRequestedStateStr**
  Description: Gets Requested State as string
  Syntax: string getRequestedStateStr (void) const;
  Returns: The Requested State as string

- **getConnectionMode**
  Description: Gets the connection mode
  Syntax: string getConnectionMode (void) const;
  Returns: The connection mode

- **getResetTimeout**
  Description: Gets ResetTimeout
  Syntax: datetime getResetTimeout (void) const;
  Returns: The resettimeout value

- **getSessionTimeout**
  Description: Gets SessionTimeout
  Syntax: datetime getSessionTimeout (void) const;
  Returns: The sessiontimeout value

- **capableOfUSBRServiceRequestStateChange**
  Description: Verifies whether the USBRService RequestStateChange operation is supported or not.
  Syntax: ***bool capableOfUSBRServiceRequestedStateChange (void) ;***
  Returns: True or False

- **capableOfUSBRServiceRequestStateChange**
  Description: Verifies whether the USBRService RequestStateChange operation is supported or not.
  Syntax: *bool* ***capableOfUSBRServiceRequestedStateChange (CIM_USBRedirectionService &te) ;***
  Parameters:
    - *CIM_USBRedirectionService* Output parameter passed as reference.

  Returns: True or False

- **capableOfUSBRSAPRequestStateChange**
  Description: Verifies whether the USBRSAP RequestStateChange operation is

supported or not.

Syntax: ***bool capableOfUSBRSAPRequestedStateChange (void) ;***

Returns: True or False

- **capableOfUSBRSAPRequestStateChange**

  Description: Verifies whether the USBRSAP RequestStateChange operation is supported or not.

  Syntax: *bool* ***capableOfUSBRSAPRequestedStateChange (CIM_USBRedirectionService &te) ;***

  Parameters:

  - *CIM_USBRedirectionService* Output parameter passed as reference.

  Returns: True or False

- **isSupportedRequestedState**

  Description: Verifies whether the perticular Requested States property is supported or not.

  Syntax: *bool* ***isSupportedReuestedState (CIM_USBRedirectionService &te, uint16 val) const ;***

  Parameters:

  - *CIM_USBRedirectionService* Output parameter passed as reference.
  - *Uint16* Integer value of RequestedState.

  Returns: True or False

- **getSupportedStates**

  Description: Gets the SupportedStates values as Integer vector.

  Syntax: *bool****igetSupportedStates (vector<uint16> &val) const ;***

  Parameters:

  - *Uint16* Integer vectore passed as reference to return value of SupportedStates.

  Returns: True or False

- **getSupportedStatesStr**

  Description: Gets the SupportedStates values as String vector.

  Syntax: *bool****igetSupportedStatesStr (vector<string> &val) const ;***

  Parameters:

  - *Uint16* String vectore passed as reference to return value of SupportedStates.

  Returns: True or False

- **activate**

  Description: Enable/Activate this redirection session

Syntax: void activate (void) const;
Returns: None

- **enable**
  Description: Enable this redirection session.
  Syntax: void enable (void) const;
  Returns: None

- **disable**
  Description: Gets the Name.
  Syntax: Disable this redirection session
  Returns: None

- **startFolderRedirection**
  Description: Starts the NFS/Samba share redirection.
  Syntax: void startFolderRedirection (string path) const;
  Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- usbredirection.h
Library -- dashapi

3.3.24. CVirtualMedia

This a class that represents VirtualMedia

Public Functions

- CVirtualMedia
- getLocalDrives
- startRedirection
- stopRerdirection

Member Functions Description

- **CVirtualMedia**
  Description: Constructs a Virtual Media Object
  Syntax: CvirtualMedia(string host,int port,string user,string password,
  string name,RedirectionTYPE_E = DRIVE,
  bool secure = false,bool write_support = false);
  Parameters:
  - *host* host ip
  - *port* Port number
  - *user* User Name
  - *password* Password
  - *RedirectionTYPE_E* DRIVE or IMAGE or FOLDER
  - *secure* false
  - *write_support* false

- **getLocalDrives**
  Description: Gets the LocalDrives
  Syntax: vector< string > getLocalDrives (void);
  Returns: List of Local Drives.

- **startRedirection**
  Description: Starts the Redirection
  Syntax: int startRedirection (void)

- **stopRedirection**
  Description: Stops the Redirection
  Syntax: int stopRedirection (void)

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- virtualmedia.h
Library -- dashapi

3.3.25 CEthernernetPort

This a class that represents EthernetPort

Public Functions

- getPortType

- getPortTypeStr
- getNetworkAddresses
- getCapabilities
- getCapabilitiesStr
- getEnabledCapabilities
- getEnabledCapabilitiesStr
- getLinkTechnology
- getPermanentAddress
- getDeviceID

Static Member Functions

- enumEthernetPorts
- getCachedProps

Constructor Description

- **CEthernetPort**

  Description: Constructs this object from the corresponding CIM_EthernetPort object.

  Syntax: CEthernetPort (const CIM_EthernetPort& eth);

  Parameters:

  - *eth* CIM EthernetPort object.

Member Functions Description

- **enumEthernetPorts**

  Description: Enumerates all the EthernetPorts present under a management access point.

  Syntax: CEthernetPort::iterator enumEthernetPorts (IClient* client, bool cached = true);

  Parameters:

  - *client* Pointer to the client interface.
  - *Cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the EthernetPort.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_EthernetPort* getCIMObject (void);

  Returns: The underlying CIM object

- **getPortType**
  Description: Gets PortType
  Syntax: uint16 getPortType (void) const;
  Returns: The type of the port

- **getPortTypeStr**
  Description: Gets port type as string
  Syntax: string getPortTypeStr (void) const;
  Returns: The port type as string

- **getNetworkAddresses**
  Description: Gets an array of NetworkAddresses
  Syntax: vector<string> getNetworkAddresses (void) const;
  Returns: The list of network addresses

- **getCapabilities**
  Description: Gets an array of Capabilities
  Syntax: vector<uint16> getCapabilities (void) const;
  Returns: The capabilities

- **getCapabilitiesStr**
  Description: Gets the capabilities as string
  Syntax: vector<string> getCapabilitiesStr (void) const;
  Returns: The capabilities as string

- **getEnabledCapabilities**
  Description: Gets an array of EnabledCapabilities
  Syntax: vector<uint16> getEnabledCapabilities (void) const;
  Returns: The enabled capabilities

- **getEnabledCapabilitiesStr**
  Description: Gets Enabled Capabalities as string
  Syntax: vector<string> getEnabledCapabilitiesStr (void) const;
  Returns: The enabled capabilities as string

- **getLinkTechnology**
  Description: Gets LinkTechnology
  Syntax: uint16 getLinkTechnology (void) const;
  Returns: The Link Technology

- **getPermanentAddress**
  Description: Gets PermanentAddress

Syntax: string getPermanentAddress (void) const;
Returns: The permanant address

- **getDeviceID**
  Description: Gets DeviceID
  Syntax: string getDeviceID (void) const;
  Returns: The DeviceID

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- ethernetport.h
Library -- dashapi

3.3.26. CRegisteredProfile

This class represents RegisteredProfile.

Public Functions

- getRegisteredName
- getRegisteredOrganization
- getValueRegisteredOrganizationStr
- getRegisteredVersion
- getAdvertiseTypes
- getValueAdvertiseTypesStr
- getInstanceID

Static Member Functions

- enumRegisteredProfile
- getCachedProps

Constructor Description

- **CRegisteredProfile**
  Description: Constructs this object from the corresponding CIM_RegisteredProfile object.

Syntax: CRegisteredProfile (const CIM_RegisteredProfile& rp);

Parameters:

- *rp* CIM_RegisteredProfile object

Member Functions Description

- **enumRegisteredProfile**

  Description: Enumerates all the RegisteredProfile present under a management access point.

  Syntax: CRegisteredProfile::iterator enumRegisteredProfile (IClient* client, bool cached = true);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the RegisteredProfile.

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_RegisteredProfile* getCIMObject (void);

  Returns: The underlying CIM object

- **getRegisteredName**

  Description: Gets the RegisteredName.

  Syntax: string getRegisteredName (void) ;

  Returns: The RegisteredName

- **getRegisteredOrganization**

  Description: Gets the Connection Mode

  Syntax: uint16 getRegisteredOrganization (void) ;

  Returns: The Registered organization

- **getValueRegisteredOrganizationStr**

  Description: Gets RegisteredOrganization as string

  Syntax: string getValueRegisteredOrganizationStr (void);

  Returns: The registered organization as string

- **getRegisteredVersion**

  Description: Gets RegisteredVersion.

  Syntax: string getRegisteredVersion (void);

Returns: The Registered version

- **getAdvertiseTypes**
  Description: Gets an array of AdvertiseTypes
  Syntax: vector<uint16> getAdvertiseTypes (void);
  Returns: The advertise types

- **getValueAdvertiseTypesStr**
  Description: Gets an array of AdvertiseTypes as strings
  Syntax: vector<string> getValueAdvertiseTypesStr (void);
  Returns: The advertise types as strings

- **getInstanceID**
  Description: Gets InstanceID
  Syntax: string getInstanceID (void);
  Returns: The InstanceID

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError
- EFunctionReturnedWithFailure

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- registreedprofile.h
Library -- dashapi

3.3.27. CIndicationFilter

This a class that represents IndicationFilter.

Public Functions

- getName
- getQuery
- getQueryLanguage
- getIndividualSubscriptionSupported
- getCreationClassName
- getSystemName
- getFilterCreationEnabled
- deleteFilter

- createFilter

Static Member Functions

- enumIndicationFilter
- getCachedProps

Constructor Description

- **CIndicationFilter**

  Description: Constructs this object from the corresponding CIM_IndicationFilter object.

  Syntax: CIndicationFilter::CIndicationFilter (const CIndicationFilter& If)

  Parameters:

  - *if* CIM_IndicationFilter object.

Member Functions Description

- **enumIndicationFilter**

  Description: Enumerates all the IndicationFilter present under a management access point.

  Syntax: CIndicationFilter::enumIndicationFilter (IClient* client, bool cached)

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the IndicationFilter

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **getCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_IndicationFilter* getCIMObject (void) const { return _if; }

  Returns: The underlying CIM object

- **getName**

  Description: Gets the Name.

  Syntax: string getName (void) const;

  Returns: The Name

- **getQuery**

  Description: Gets the Query stirng for this filter

  Syntax: string getQuery (void) const;

Returns: The Query

- **getQueryLanguage**
  Description: Gets the query language
  Syntax: string getQueryLanguage (void) const;
  Returns: The query language

- **getIndividualSubscriptionSupported**
  Description: Gets IndividualSubscriptionSupported
  Syntax: boolean getIndividualSubscriptionSupported (void) const;
  Returns: true if supported
  false otherwise

- **getCreationClassName**
  Description: Gets the CreationClassName
  Syntax: string getCreationClassName (void) const;
  Returns: The Creation class name

- **getSystemName**
  Description: Gets the SystemName
  Syntax: string getSystemName (void) const;
  Returns: The SystemName

- **getFilterCreationEnabled**
  Description: Gets the FilterCreationEnabled
  Syntax: static boolean getFilterCreationEnabled (void) ;
  Returns: true if enabled
  false otherwise
- **deleteFilter**
  Description: deleteFilter
  Syntax: void deleteFilter (void) const;
  Returns: None

- **createFilter**
  Description: createFilter
  Syntax: static CIndicationFilter createFilter (IClient* client, string querylanguage, string query);
  Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- indications.h
Library -- dashapi

3.3.28. CAlertDestination

This a class that represents AlertDestination

Public Functions

- getName
- getDestination
- getProtocol
- getPersistenceType
- deleteDestination
- createDestination

Static Member Functions

- enumAlertDestination
- getCachedProps

Constructor Description

- **CAlertDestination**

  Description: Constructs this object from the corresponding CIM_ListenerDestination object.

  Syntax: CAlertDestination::CAlertDestination (const CIM_ListenerDestination& ld)

  Parameters:

  - *ld* CIM_ListenerDestination object.

Member Functions Description

- **enumAlertDestination**

  Description: Enumerates all the AlertDestination present under a management access point.

  Syntax: CAlertDestination::enumAlertDestination (IClient* client, bool cached)

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the AlertDestination

- **getCachedProps**

  Description: Gets the properties that are cached by this object

Syntax: vector< string > getCachedProps (void);
Returns: List of cached properties.

- **getCIMObject**
  Description: Gets the underlying CIM object
  Syntax: CIM_ListenerDestination* getCIMObject (void) const { return _ld; }
  Returns: The underlying CIM object

- **getName**
  Description: Gets the Name.
  Syntax: string getName (void) const;
  Returns: The Name

- **getDestination**
  Description: Gets the destination of the alert
  Syntax: string getDestination (void) const;
  Returns: The destination

- **getProtocol**
  Description: Gets the Protocol
  Syntax: uint16 getProtocol (void) const;
  Returns: The listener destination protocol

- **getPersistenceType**
  Description: Gets PersistenceType
  Syntax: uint16 getPersistenceType (void) const;
  Returns: The persistance type

- **deleteDestination**
  Description: Deletes this alert destination
  Syntax: void deleteDestination (void) const;

- **createDestination**
  Description: Create a Alert destination
  Syntax: static CAlertDestination createDestination (IClient* client, string destination);

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions.

Class Requirements

Header file -- indications.h
Library -- dashapi

3.3.29. CAbstractIndicationSubscription

This a class that represents AbstractIndicationSubscription

Public Functions

- getOnFatalErrorPolicy
- getOtherOnFatalErrorPolicy
- getFailureTriggerTimeInterval
- getSubscriptionState
- getOtherSubscriptionState
- getRepeatNotificationPolicy
- getRepeatNotificationInterval
- getRepeatNotificationGap
- getRepeatNotificationCount
- getFilternName
- getAlertDestination
- getFilter
- getHandler
- unSubscribe
- renewSubscription

Static Member Functions

- enumIndicationSubscription
- getCachedProps

Constructor Description

- **CIndicationSubscription**

  Description: Constructs this object from the corresponding CIM_IndicationSubscription object.

  Syntax: CAbstractIndicationSubscription (const CAbstractIndicationSubscription& _ais)

  Parameters:

  - *ais* CIM_ AbstractIndicationSubscription object.

Member Functions Description

- **enumAbstractIndicationSubscription**

  Description: Enumerates all the AbstractIndicationSubscription present under a management access point.

  Syntax: CAbstractIndicationSubscription::iterator enumIndicationSubscription (IClient* client, bool cached)

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching. Default is true.

Returns: Iterator to the IndicationSubscription

- **getCachedProps**

  Description: Gets the properties that are cached by this object
  Syntax: vector< string > getCachedProps (void);
  Returns: List of cached properties.

- **GetCIMObject**

  Description: Gets the underlying CIM object
  Syntax: CIM_AbstractIndicationSubscription<CIM_ManagedElement, CIM_ListenerDestination>* getCIMObject (void) const { return _ais; }
  Returns: The underlying CIM object

- **getOnFatalErrorPolicy**

  Description: Gets OnFatalErrorPolicy
  Syntax: uint16 getOnFatalErrorPolicy (void) const;
  Returns: The OnFatalErrorPolicy value

- **getOtherOnFatalErrorPolicy**

  Description: Gets OtherOnFatalErrorPolicy string
  Syntax: string getOtherOnFatalErrorPolicy (void) const;
  Returns: The OtherOnFatalErrorPolicy string

- **getFailureTriggerTimeInterval**

  Description: Gets Failure Trigger TimeInterval
  Syntax: uint64 getFailureTriggerTimeInterval (void) const;
  Returns: The FailureTriggerTimeInterval

- **getSubscriptionState**

  Description: Gets SubscriptionState value
  Syntax: uint16 getSubscriptionState (void) const;
  Returns: The SubscriptionState value

- **getOtherSubscriptionState**

  Description: Gets OtherSubscriptionState string
  Syntax: string getOtherSubscriptionState (void) const;
  Returns: The OtherSubscriptionState string

- **getRepeatNotificationPolicy**

  Description: Gets RepeatNotificationPolicy

Syntax: uint16 getRepeatNotificationPolicy (void) const;
Returns: The RepeatNotificationPolicy value

- **getRepeatNotificationInterval**
  Description: Gets RepeatNotification Interval
  Syntax: uint64 getRepeatNotificationInterval (void) const;
  Returns: The RepeatNotificationInterval

- **getRepeatNotificationGap**
  Description: Gets RepeatNotificationGap
  Syntax: uint64 getRepeatNotificationGap (void) const;
  Returns: The RepeatNotificationGap value

- **getRepeatNotificationCount**
  Description: Gets RepeatNotification Count
  Syntax: uint16 getRepeatNotificationCount (void) const;
  Returns: The RepeatNotificationCount

- **getFilternName**
  Description: Gets the filter name
  Syntax: string getFilternName (IClient* client) const;
  Parameter:
- client Pointer to the client interface
  Returns: The filter name string

- **getAlertDestination**
  Description: Gets the AlertDestination string
  Syntax: string getAlertDestination (IClient* client) const;
  Parameter:
  - client Pointer to the client interface
  Returns: The alert destination string

- **getFilter**
  Description: The Filter that defines the criteria and data of the possible Indications of this subscription.
  Syntax: CCIMObjectPath getFilter(void) const;
  Returns: The filter that defines the criteria and data of the possible Indications

- **getHandler**
  Description: The Handler addressing delivery of the possible Indications of this subscription.
  Syntax: CCIMObjectPath getHandler(void) const;

Returns: The Handler addressing delivery of the possible Indications

- **unSubscribe**

  Description: Unsubcribe this Subscription
  Syntax: void unSubscribe (void) const;
  Returns: none

- **renewSubscription**

  Description: Renews the given subscription
  Syntax: void renewSubscription (string renewtime) const;
  Returns: none
  Paramerters:
- renewtime: renewal time

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- indications.h
Library -- dashapi

3.3.30. CIndicationSubscription

This a class that represents IndicationSubscription

Public Functions

- getOnFatalErrorPolicy
- getOtherOnFatalErrorPolicy
- getFailureTriggerTimeInterval
- getSubscriptionState
- getOtherSubscriptionState
- getRepeatNotificationPolicy
- getRepeatNotificationInterval
- getRepeatNotificationGap
- getRepeatNotificationCount
- getFilternName
- getAlertDestination
- getFilter
- getHandler
- deleteSubscription

- createSubscription

Static Member Functions

- enumIndicationSubscription
- getCachedProps

Constructor Description

- **CIndicationSubscription**

  Description: Constructs this object from the corresponding CIM_IndicationSubscription object.

  Syntax: CIndicationSubscription::CIndicationSubscription (const CIndicationSubscription& is)

  Parameters:

  - *is* CIM_ IndicationSubscription object.

Member Functions Description

- **enumIndicationSubscription**

  Description: Enumerates all the IndicationSubscription present under a management access point.

  Syntax: CindicationSubscription::enumIndicationSubscription (IClient* client, bool cached)

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the IndicationSubscription

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **GetCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_IndicationSubscription<CIM_IndicationFilter, CIM_ListenerDestination>* getCIMObject (void) const { return _ais; }

  Returns: The underlying CIM object

- **getOnFatalErrorPolicy**

  Description: Gets OnFatalErrorPolicy

  Syntax: uint16 getOnFatalErrorPolicy (void) const;

  Returns: The OnFatalErrorPolicy value

- **getOtherOnFatalErrorPolicy**

Description: Gets OtherOnFatalErrorPolicy string
Syntax: string getOtherOnFatalErrorPolicy (void) const;
Returns: The OtherOnFatalErrorPolicy string

- **getFailureTriggerTimeInterval**
  Description: Gets Failure Trigger TimeInterval
  Syntax: uint64 getFailureTriggerTimeInterval (void) const;
  Returns: The FailureTriggerTimeInterval

- **getSubscriptionState**
  Description: Gets SubscriptionState value
  Syntax: uint16 getSubscriptionState (void) const;
  Returns: The SubscriptionState value

- **getOtherSubscriptionState**
  Description: Gets OtherSubscriptionState string
  Syntax: string getOtherSubscriptionState (void) const;
  Returns: The OtherSubscriptionState string

- **getRepeatNotificationPolicy**
  Description: Gets RepeatNotificationPolicy
  Syntax: uint16 getRepeatNotificationPolicy (void) const;
  Returns: The RepeatNotificationPolicy value

- **getRepeatNotificationInterval**
  Description: Gets RepeatNotification Interval
  Syntax: uint64 getRepeatNotificationInterval (void) const;
  Returns: The RepeatNotificationInterval

- **getRepeatNotificationGap**
  Description: Gets RepeatNotificationGap
  Syntax: uint64 getRepeatNotificationGap (void) const;
  Returns: The RepeatNotificationGap value

- **getRepeatNotificationCount**
  Description: Gets RepeatNotification Count
  Syntax: uint16 getRepeatNotificationCount (void) const;
  Returns: The RepeatNotificationCount

- **getFilternName**
  Description: Gets the filter name
  Syntax: string getFilternName (IClient* client) const;
  Parameter:

- client Pointer to the client interface

Returns: The filter name string

- **getAlertDestination**

  Description: Gets the AlertDestination string

  Syntax: string getAlertDestination (IClient* client) const;

  Parameter:

  - client Pointer to the client interface

  Returns: The alert destination string

- **getFilter**

  Description: The Filter that defines the criteria and data of the possible Indications of this subscription.

  Syntax: CCIMObjectPath getFilter(void) const;

  Returns: The filter that defines the criteria and data of the possible Indications

- **getHandler**

  Description: The Handler addressing delivery of the possible Indications of this subscription.

  Syntax: CCIMObjectPath getHandler(void) const;

  Returns: The Handler addressing delivery of the possible Indications

- **deleteSubscription**

  Description: Deletes the subscription instance

  Syntax: void deleteSubscription (void) const;

  Returns: None

- **createSubscription**

  Description: Create a Alert/Indication subscription.

  Syntax: static string createSubscription (IClient* client,string querylanguage, string query,string destination,int mode, float heartbeat_interval, float expiration_timeout);

  Syntax: static string createSubscription (IClient* client, CIndicationFilter filter, string destination, int mode, float heartbeat_interval, float expiration_timeout);

  Parameters:

  - client Pointer to the client interface.
  - querylanguage Query language to be used for the filter.
    Example "CQL" for CIM query language.
  - query Query string to create the filter based on the querylanguage.
    Example "SELECT * FROM CIM_AlertIndication" for CQL.

- destination Destination stirng in below format
  http://<ipaddress>:<port>/path
  Example http://192.168.0.11:8080/eventsink
- mode MODE_PUSH, MODE_PUSH_ACK,MODE_PULL
- heartbeat_interval The interval in which heartbeat events are sent.
- expiration_timeout The timeout by which the subscription expires.

Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- indications.h
Library -- dashapi

3.3.31. CFilterCollectionSubscription

This a class that represents FilterCollectionSubscription

Public Functions

- getOnFatalErrorPolicy
- getOtherOnFatalErrorPolicy
- getFailureTriggerTimeInterval
- getSubscriptionState
- getOtherSubscriptionState
- getRepeatNotificationPolicy
- getRepeatNotificationInterval
- getRepeatNotificationGap
- getRepeatNotificationCount
- getFilternName
- getAlertDestination
- getFilter
- getHandler
- deleteFilterCollectionSubscription
- createFilterCollectionSubscription

Static Member Functions

- enumFilterCollectionSubscription
- getCachedProps

Constructor Description

- **CFilterCollectionSubscription**

  Description: Constructs this object from the corresponding CIM_FilterCollectionSubscription object.

  Syntax: CFilterCollectionSubscription::CFilterCollectionSubscription (const CFilterCollectionSubscription& fcs)

  Parameters:

  - *fcs* CIM_ FilterCollectionSubscription object.

Member Functions Description

- **enumFilterCollectionSubscription**

  Description: Enumerates all the FilterCollectionSubscription present under a management access point.

  Syntax: CFilterCollectionSubscription::enumFilterCollectionSubscription (IClient* client,bool cached)

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching. Default is true.

  Returns: Iterator to the FilterCollectionSubscription

- **getCachedProps**

  Description: Gets the properties that are cached by this object

  Syntax: vector< string > getCachedProps (void);

  Returns: List of cached properties.

- **GetCIMObject**

  Description: Gets the underlying CIM object

  Syntax: CIM_AbstractFilterCollectionSubscription<CIM_ManagedElement, CIM_ListenerDestination>* getCIMObject (void) const { return _is; }

  Returns: The underlying CIM object

- **getOnFatalErrorPolicy**

  Description: Gets OnFatalErrorPolicy

  Syntax: uint16 getOnFatalErrorPolicy (void) const;

  Returns: The OnFatalErrorPolicy value

- **getOtherOnFatalErrorPolicy**

  Description: Gets OtherOnFatalErrorPolicy string

  Syntax: string getOtherOnFatalErrorPolicy (void) const;

  Returns: The OtherOnFatalErrorPolicy string

- **getFailureTriggerTimeInterval**

Description: Gets Failure Trigger TimeInterval
Syntax: uint64 getFailureTriggerTimeInterval (void) const;
Returns: The FailureTriggerTimeInterval

- **getSubscriptionState**
  Description: Gets SubscriptionState value
  Syntax: uint16 getSubscriptionState (void) const;
  Returns: The SubscriptionState value

- **getOtherSubscriptionState**
  Description: Gets OtherSubscriptionState string
  Syntax: string getOtherSubscriptionState (void) const;
  Returns: The OtherSubscriptionState string

- **getRepeatNotificationPolicy**
  Description: Gets RepeatNotificationPolicy
  Syntax: uint16 getRepeatNotificationPolicy (void) const;
  Returns: The RepeatNotificationPolicy value

- **getRepeatNotificationInterval**
  Description: Gets RepeatNotification Interval
  Syntax: uint64 getRepeatNotificationInterval (void) const;
  Returns: The RepeatNotificationInterval

- **getRepeatNotificationGap**
  Description: Gets RepeatNotificationGap
  Syntax: uint64 getRepeatNotificationGap (void) const;
  Returns: The RepeatNotificationGap value

- **getRepeatNotificationCount**
  Description: Gets RepeatNotification Count
  Syntax: uint16 getRepeatNotificationCount (void) const;
  Returns: The RepeatNotificationCount

- **getFilternName**
  Description: Gets the filter name
  Syntax: string getFilternName (IClient* client) const;
  Parameter:
    - client Pointer to the client interface
  Returns: The filter name string

- **getAlertDestination**
  Description: Gets the AlertDestination string

Syntax: string getAlertDestination (IClient* client) const;

Parameter:

- client Pointer to the client interface

Returns: The alert destination string

- **getFilter**

  Description: The Filter that defines the criteria and data of the possible Indications of this subscription.

  Syntax: CCIMObjectPath getFilter(void) const;

  Returns: The filter that defines the criteria and data of the possible Indications

- **getHandler**

  Description: The Handler addressing delivery of the possible FilterCollectionof this subscription.

  Syntax: CCIMObjectPath getHandler(void) const;

  Returns: The Handler addressing delivery of the possible FilterCollection

- **deleteFilterCollectionSubscription**

  Description: Deletes the subscription instance

  Syntax: void deleteSubscription (void) const;

  Returns: None

- **createFilterCollectionSubscription**

  Description: Create a Alert/Indication subscription.

  Syntax: static CFilterCollectionSubscription createSubscription (IClient* client, CFilterCollection filter, CAlertDestination destination);

  Syntax: static string createSubscription (IClient* client,
  CFilterCollection filtercollection,
  string destination,
  int mode,
  float heartbeat_interval,
  float expiration_timeout);

  Parameters:

  - client Pointer to the client interface.
  - filtercollection Filtercollection to be used for creating subscriptoin.
  - destination Destination stirng in below format
    http://<ipaddress>:<port>/path
    Example http://192.168.0.11:8080/eventsink
  - mode MODE_PUSH, MODE_PUSH_ACK, MODE_PULL
  - heartbeat_interval The interval in which heartbeat events are sent.
  - expiration_timeout The timeout by which the subscription expire.

  Returns: None

Note: All the Member function in this class throws exceptions if error occurs. Following exceptions are thrown by this API.

- EDSDKError.
- ECIMError

The application needs to handle these exception. Refer section 3.4 for details on exceptions

Class Requirements

Header file -- filtercollection.h
Library -- dashapi

3.4 Exceptions

3.4.1 DASH SDK Exceptions

EDSDKError

A generic DASH SDK Error Exception. All the exceptions thrown by the DASH SDK are derived from this exception. EDSDKError is derived from the c++ exception class.

Member Functions:

Functions member of this class consist of the following:

- EDSDKError : This is constructor.
- what
- getErrorCode

Constructor Description:

- **EDSDKError**
  Construct this object from the error code and message string.

  To Construct this object from the error code.
  Syntax: EDSDKError (int error_code)
  Parameters:
  - error_code Error code

  To Construct this object from the error code and message string.
  Syntax: EDSDKError (int error_code, string str)
  Parameters:
  - *error_code* Error code
  - *str* Message string

Member Functions Description

- **what**

  Description : This function Describes this exception
  Syntax: virtual const char* what (void) const throw ()
  Returns: The error string.

- **getErrorCode**

  Description : This function gets the return error code.
  Syntax: virtual unsigned int getErrorCode (void) const throw ()
  Returns : The error code

- **ECIMError:**
  It is derived from EDSDKError.A CIM error exception throws when there is a CIM error

Member Functions:

Functions member of this class consist of the following:

- ECIMError . This is constructor.
- checkThrowCIMStatus
- what
- getErrorCode
- getCIMErrorMsg

Constructor Description:

- **ECIMError**
  Build this class from the low level status code
  Syntax : ECIMError (const CMPIStatus& status):EDSDKError(DSDK_CIM_ERROR)
  Parameters:
  - *status* status message

Member Functions Description

- **checkThrowCIMStatus**
  Description: Static inline function that checks if the status has error and throws the exception.
  Syntax: static inline void checkThrowCIMStatus (const CMPIStatus& status)
  Parameter:
  - *status* The Status

- **what**

Description: This function Describes this exception

Syntax: virtual const char* what (void) const throw ()

Returns : The error string.

- **GetErrorCode**

  Description : This function gets the return error code.

  Syntax: virtual unsigned int getErrorCode (void) const throw ()

  Returns : The error code

- **getCIMErrorMsg**

  Description: This function gets CIM Error message

  Syntax: virtual string getCIMErrorMsg (void)

  Returns: The Error code

- **ECIMMethodNotImplemented:**

  Method not implemented macro. This exception is thrown when unsupported CIM method is called. Call getMethod function to get the method name.

Member Functions:

Functions member of this class consist of the following:

- ECIMMethodNotImplemented : This is constructor.
- what
- getMethod
- checkThrowCIMMethodNotImplemented

Constructor Description:

- **ECIMMethodNotImplemented**

  Static inline function that checks if the status has method not implemented and throws the exception.

  Syntax : ECIMMethodNotImplemented (string name) : EDSDKError (DSDK_CIMMETHOD_NOTIMPLEMENTED), _name (name)

  Parameters:

  - *name* The name

Member Functions Description

- **what**

  Description : This function Describes this exception

  Syntax: virtual const char* what (void) const throw ()

  Returns: The error string.

- **getMethod**

Description : This function gets the return code.

Syntax: virtual string getMethod (void) const throw ()

Returns : The name

- **checkThrowCIMMethodNotImplemented**

  Description : This function gets the return code.

  Syntax: static inline void checkThrowCIMMethodNotImplemented (const CMPIStatus& status, string name)

  Parameters:

  - *status* The status
  - *name* The name

- **EconnectionFailed:**

  A connection failed exception. This exception is thrown when a connection error like when not able to connect to the server or not able to authenticate. Call getconnectionFailedMsg to get the error message.

Member Functions:

Functions member of this class consist of the following:

- EconnectionFailed : This is constructor.
- what
- getConnectionFailedMsg
- checkThrowConnectionFailed

Constructor Description:

- **EconnectionFailed**

  Build this class from the low level status code

  Syntax: EConnectionFailed (const CMPIStatus& status) : EDSDKError (DSDK_CONNECTION_FAILED)

  Parameters:

  - *status* The status

Member Functions Description

- **what**

  Description : This function Describes this exception

  Syntax: virtual const char* what (void) const throw ()

  Returns: The error string.

- **getConnectionFailedMsg**

  Description : This function gets the connection failed error message

Syntax: virtual string getConnectionFailedMsg (void)

Returns : The connection failed error message

- **checkThrowConnectionFailed**

  Description : This function gets the connection failed error message

  Syntax: static inline void checkThrowConnectionFailed (const CMPIStatus& status)

  Parameters:
  - *status* The status

- **EnotEnoughMemory:**

  Not enough memory exception.

Member Functions:

Functions member of this class consist of the following:
- EnotEnoughMemory : This is constructor.

Constructor Description:

- **EnotEnoughMemory**

  Not enough memory Constructor

  Syntax : ENotEnoughMemory () : EDSDKError (DSDK_NOT_ENOUGH_MEMORY)

- **EFunctionNotSupported**

  Function is not supported by the remote

Member Functions:

Functions member of this class consist of the following:
- EFunctionNotSupported : This is constructor.
- what
- getFunction

Constructor Description:

- **EFunctionNotSupported**

  Function not supported constructor.

  Syntax : EFunctionNotSupported (string function) : EDSDKError (DSDK_FUNCTION_NOT_SUPPORTED)

  Parameter:
  - *function* The function name

Member Functions Description

- **what**

  Description : This function Describes this exception

  Syntax: virtual const char* what (void) const throw ()

  Returns: The error string.

- **getFunction**

  Description : This function gets the name of function not supported

  Syntax: virtual string getFunction (void)

  Returns : The name of function

- **EfunctionReturnedWithFailure**

  A brief Function is returned with failure. This execption is thrown when a CIM Extrinsic method is returned with failure. Call getFunction to get thefunction name and call getErrorMsg to get the failure message.

Member Functions:

Functions member of this class consist of the following:

- EfunctionReturnedWithFailure : This is constructor.
- What
- getFunction
- getErrorMsg
- getRetCode

Constructor Description:

- **EfunctionReturnedWithFailure**

  Function returned with failure constructor.

  Syntax : EFunctionReturnedWithFailure (string fnname, string retcodestr,uint32 status):EDSDKError (DSDK_FUNCTION_RETURNED_WITH_FAILURE)

  Parameters:

  - *fnname* Function Name
  - *retcodestr* Returned Code as string
  - *status* The status

Member Functions Description

- **what**

  Description : This function Describes this exception

  Syntax: virtual const char* what (void) const throw ()

  Returns: The error string.

- **getFunction**
  Description : This function gets the name of function not supported
  Syntax: virtual string getFunction (void)
  Returns : The name of function

- **getErrorMsg**
  Description : This function gets the retcode string of function
  Syntax: virtual string getErrorMsg (void)
  Returns : The retcode string of function

- **getRetCode**
  Description : This function gets the return code
  Syntax: virtual unsigned int getRetCode (void)
  Returns : The status value

3.5 Use Case

3.5.1 Profiles supported/Advertised:

1)How to get the list of profiles supported/advertised ?

To get the list of profiles advertised by a MAP.

1. Enumerate the registered profile using static function CregisteredProfile::enumerateRegisteredProfile
2. From the list of enumerate profiles get the name of the profiles using getRegisteredName ().

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "registeredprofile.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
```

```
        try
        {
                /* enumerate the registered profile */
                CRegisteredProfile::iterator i = CRegisteredProfile::
                                enumRegisteredProfile (client);
                for (; i != CRegisteredProfile::iterator::end (); ++i)
                {
                        /* get the instances of the registered profiles */
                        CRegisteredProfile rp = *i;

                        /* get the names of the registered profiles */
                        try
                        {
                            fprintf(stdout, "Name  :s\n", rp.getRegisteredName ());
                        }
                        catch ( EDSDKError &e)
                        {
                            fprintf (stdout, "Error getting registered name :   %s\n", e.what());
                        }
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing registered profiles : %s\n",
                    e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Name  : Battery Profile
Name  : Fan Profile
Name  : Boot Control Profile
Name  : CPU Profile
Name  : OS Status Profile
Name  : System Memory Profile
Name  : Physical Asset Profile
Name  : Power Supply Profile
Name  : Sensors Profile
Name  : Software Inventory Profile
Name  : Software Update Profile
Name  : USB Redirection Profile
```

3.5.2 Check for a profile supported/advertised.

1)How to check if a profile is supported or advertised?

To check if a profile is supported/ advertised in MAP

1. Call the CRegisteredProfile::isAdvertised, with the profile name to check.

Check the User Profile is supported

```
#include <iostream.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "registeredprofile.h"

using namespace dsdk;

int
main (void)
{
        string profilename;
        cout<<"Enter the profile name\n";
        cin>>profilename;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the registered profile */
                if (CRegisteredProfile::isAdvertised (client, profilename))
                {
                        fprintf (stdout, "The profile %s is  advertised/supported\n", profilename);
                }
                else
                {
                         fprintf (stdout, "The profile %s is not advertised/supported\n", profilename);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing registered profile : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the profile name
```

```
Fan Profile
The profile Fan Profile is advertised/supported

Enter the profile name
Batery Profile
The profile Batery Profile is not advertised/supported
```

3.5.3 Access Profile properties.

1) How to get a properties of particular instance of a profile.?

1. Enumerate a profile by calling the static enumerate function of profile API.
2. Get the particular instance of the profile using the iterator.
3. Get the properties of the instance using the getter functions.

Note: The enumeration will start from the instance 0.

2) How to get the Computer owner of Computer System Named "Managed System"?

To get the list of computerststem advertised by a MAP.

1. Enumerate the computersystem using static function CComputerSystem::enumerateComputersystem
2. From the list of computer system get the name of the computersystem named "Managed System".

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::enumComputerSystem (client);

                /* get the instances of all the computersystem */
                for (; i != CComputerSystem::iterator::end (); ++i)
                {
                        CComputerSystem cs = *i;
```

```
                        /* get the names of the computersystem and check that to Managed System*/
                        if (cs.getName == "Managed System")
                        {
                                fprintf (stdout, "Primary Owner : %s\n", cs.getPrimaryOwner().c_str ());
                                return 0;
                        }
                }
                fprintf (stdout, "Computer system with name "Managed System" not found\n");
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computersystem : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Primary Owner : Raritan
```

3) How to "power on" a particular computer system?

1. Enumerate all the instance of ComputerSystem using ComputerSystem::enumComputerSystem
2. Find the name of the computer system to power on.
3. Invoke the power on method of this instance of computer system.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
main (void)
{
        string sysName;
        cout <<"Enter the name of the system to be power on\n";
        cin >>sysName;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
```

```
        {
                /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::enumComputerSystem (client);
                /* get the instances of the computersystem */
                for (; i != CComputerSystem::iterator::end (); ++i)
                {
                        CComputerSystem cs = *i;

                        /* find the computer system and power on */
                        if (cs.getName () == sysName)
                        {
                                if (cs.powerOn () == 0)
                                {
                                    fprintf (stdout, "Power on success\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Power on failed\n");
                                    return 0;
                                }
                        }
                }
                else
                {
                     fprintf (stdout, "Computersystem name %s not available\n",sysName);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computersystem : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the name of the system to be power on
mkl_desktop
Power on success

Enter the name of the system to be power on
xyz
Computersystem name xyz not available
```

4)How to “power off” a particular computer system?

1. Enumerate all the instance of ComputerSystem using

ComputerSystem::enumComputerSystems

2. Find the instance of the computer system to power on.
3. Invoke the power off method of this instance of computer system.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
using namespace dsdk;
int
main (void)
{
        int instance;
        cout <<"Enter the instance of the system to be power off\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
        /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::enumComputerSystem (client);
        /* get the instances of the computersystem and j value will get increment for each computersystem */
                for (int j = 0; i != CComputerSystem::iterator::end (); ++i,j++)
                {
                        CComputerSystem cs = *i;

                        /* find the computer system and power off */
                        if (j == instance)
                        {
                                if (cs.powerOff () == 0)
                                {
                                    fprintf (stdout, "Power off success\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Power off failed\n");
                                    return 0;
                                }
                        }
                }
                if (j < instance)
                {
                     fprintf (stdout, "Computersystem instance %d not available\n",instance);
                }
```

```
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computersystem : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of the system to be power off
3
Power off success

Enter the instance of the system to be power off
8
Computersystem instance 8 not available
```

5) How to do a "power reset" a particular computer system?

1. Enumerate all the instance of ComputerSystem using ComputerSystem::enumComputerSystems
2. Find the name of the computer system to power on.
3. Invoke the power reset method of this instance of computer system.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"

using namespace dsdk;

int
main (void)
{
        string sysName;
        cout <<"Enter the name of the system to power reset\n";
        cin >>sysName;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::
```

```
                        enumComputerSystem (client);
                /* get the instances of the computersystem and j value will get increment for each
                                                                                computersystem */
                for (; i != CComputerSystem::iterator::end (); ++i)
                {
                        CComputerSystem cs = *i;

                        /* find the computer system and power off */
                        if (cs.getName () == sysName)
                        {
                                if (cs.powerOff () == 0)
                                {
                                  fprintf (stdout, "Power reset success\n");
                                  return 0;
                                }
                                else
                                {
                                  fpritnf (stdout, "Power reset failed\n");
                                  return 0;
                                }
                        }
                }
                else
                {
                     fprintf (stdout, "Computersystem instance %d not available\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computersystem : %s\n", e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the name of the system to power reset
xyz
Power reset success

Enter the name of the system to power reset
abc
Computersystem name abc not available
```

6)How to add user?

1. The user can be add to the particular computersystem only.
2. Get the instance of the computersystem to add user
3. To do that, enumerate the computer system using

Ccomputersystem::enumComputerSystems.

4. Then the user can be added for the instance of the computersystem by calling createUser function from CUser class as follows.

   Cuser::createUser.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"
using namespace dsdk;
int
main (void)
{
        int instance;
        string userName, password;
        cout <<"Enter the instance of computersystem, user name & password to add user\n";
        cin >>instance>>userName>>password;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::enumComputerSystem (client);
                /* get the instances of the computersystem and j value will get increment for each
                                                                                computersystem */
                for (int j = 0; i != CComputerSystem::iterator::end (); ++i,j++)
                {
                  CComputerSystem cs = *i;
                  /* find instance of computersystem to add user */
                  if (j == instance)
                   {
                        try
                        {
                                CUser::createUser (cs, userName, password);
                                fprintf (stdout, "User Added Successfully\n");
                                return 0;
                        }
                        catch (exception &e)
                        {
                                fprintf (stderr, "Error: Adding user failed\n");
                        }
                        return 0;
                   }
                }
                if (j < instance)
```

```
                {
                    fprintf (stdout, "Computersystem instance %d not found",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computer system : %s\n", e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of computersystem, user name & password to add user
1       testuser        testpassword
User Added Successfully
```

7) How to "delete" a user?

1. The user can be delete .
2. Get the instance of the user to delete a user
3. To do that, enumerate the user using CUser::enumUsers.
4. Then the user can be deleted by calling deleteUser function from CUser class as follows.

   CUser::deleteUser.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of user to delete user\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
```

```
        {
                /* enumerate the user */
                CUser::iterator i = CUser::enumUsers (client);
                /* get the instances of the user and j value will get increment for each user */
                for (int j = 0; i != CUser::iterator::end (); ++i,j++)
                {
                  CUser user = *i;
                  /* find instance of user to delete user */
                  if (j == instance)
                   {
                        if (user.deleteUser () == 0)
                          {
                                fprintf (stdout, "User deletedsuccessfully\n");
                                return 0;
                          }
                        else
                          {
                                fpritnf (stdout, "Error: Deleting User\n");
                                return 0;
                          }
                   }
                }

                if (j < instance)
                 {
                     fprintf (stdout, "User instance %d is not found",instance);
                 }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing user : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of user to delete user
2
User deleted Successfully
Enter the instance of user to delete user
14
User instance 14 is not found
```

8) How to add role to a particular computerstystem?

1. The role can be add to the particular computersystem only.
2. Get the instance of the computersystem to add role

3. To do that, enumerate the computer system using Ccomputersystem::enumComputerSystems.
4. Then by giving rolename & activities,qualifiers(activity should be one of these\n create|delete|detect|read|write|execute|other. Qualifiers should be specified separated by commas).
5. Then the role can be added for the instance of the computersystem by calling createRole function from CRole class as follows.

   Cuser::createRole.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        string _roleName, _permissions;
        cout <<"Enter the instance of computersystem, role name & permission, to add role\n";
        cin >>_roleName>>_permissions;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the computer system */
                CComputerSystem::iterator i = CComputerSystem::enumComputerSystem (client);
                /* get the instances of the computersystem and j value will get increment for each
                                                                        computersystem */
                for (int j = 0; i != CComputerSystem::iterator::end (); ++i,j++)
                {
                        CComputerSystem cs = *i;
                  /* find instance of computersystem to add role */
                  if (j == instance)
                    {
                        {
                           vector <CRole::Permission_T> permissions;
                           CRole::Permission_T permission;
```

```
                string privilege = _permissions;
                /* seperate the activity */
                string activity = privilege.substr (0, privilege.
                                                    find (','));
                if (activity == "create") { permission.activity = CRole::Create; }
                else if (activity == "delete")
                        { permission.activity = CRole::Delete; }
                else if (activity == "detect")
                        { permission.activity = CRole::Detect; }
                else if (activity == "read")
                        { permission.activity = CRole::Read;   }
                else if (activity == "write")
                        { permission.activity = CRole::Write;  }
                else if (activity == "execute")
                        { permission.activity = Crole::Execute;}
                else if (activity == "other")
                        { permission.activity = CRole::Other;  }
                else
                {
                  fprintf (stderr, "Invalid activity \" %s \" specified", activity.c_str ());
                  fprintf (stderr, "Should be one of these\n  create|delete|detect|read|write|execute|
                                                                                    other\n");
                  return -1;
                }
                if ((size_t)-1 == privilege.find (','))
                {
                  fprintf (stderr, "Qualifier missing, should be specified, seperated by comma in the
                                                                                    permission");
                  return -1;
                }
                permission.qualifier = privilege.substr
                        (privilege.find (',') + 1, privilege.size ());
                permissions.push_back (permission);
              }
              try
              {
                CRole::createRole (cs, roleName, permissions);
                fprintf (stdout, "Role Added Successfully\n");
                return 0;
              }
              catch (exception &e)
              {
                 fprintf (stderr, "Error: Adding Role failed\n");
              }
              return 0;
    }
}
if (j < instance)
 {
     fprintf (stdout, "Computersystem instance %d not found",instance);
```

```
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing computer system : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of computersystem, role name & permission, to add role
1
testrole
write,Clear Log

Role Added Successfully
```

9)How to "removeroles" to particular user?

1. Roles can be remove to user.
2. Get the name of the user to removeroles to particular user.
3. To do that, enumerate the user using CUser::enumUsers.
4. Then the roles can be removed by calling removeRoles function from CUser class as follows.

CUser::removeRoles .

A sample usage is show below.(to remove role to user friedrick)

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"

using namespace dsdk;

int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the user */
                CUser::iterator i = CUser::enumUsers (client);
```

```
                /* get the instances of the user */
                for (; i != CUser::iterator::end (); ++i)
                {
                  CUser user = *i;
                  /* find instance of user to remove roles */
                  if (user.getName () == "friedrick")
                   {
                        try
                        {
                                vector <string> role;
                                string rolename;
                                cout<<"Enter role to remove\n";
                                role.push_back (rolename);
                        }
                                user.removeRoles (role);
                                fprintf (stdout, "Roles removed Successfully\n");
                                return 0;
                        }
                        catch (exception &e)
                        {
                                fprintf (stderr, "Error: Removing roles failed\n");
                        }
                        return 0;
                    }
                }
                if (j < instance)
                 {
                     fprintf (stdout, "User instance %d is not found",instance);
                 }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing user : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the roles to remove
admin
Roles removed Successfully
```

10) How to "assignroles" to particular user?

1. Roles can be assign to user.
2. Get the instance of the user to assignroles to particular user

3. To do that, enumerate the user using CUser::enumUsers.
4. Then the roles can be assigned by calling assignRoles function from CUser class as follows.

   CUser::assignRoles .

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"
using namespace dsdk;
int
main (void)
{
        int instance;
        int numofrole;
        cout <<"Enter the instance of user, & num. of roles to assignroles to particular user\n";
        cin >>instance>>numofrole;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the user */
                CUser::iterator i = CUser::enumUsers (client);
                /* get the instances of the user and j value will get increment for each user */
                for (int j = 0; i != CUser::iterator::end (); ++i,j++)
                {
                  CUser user = *i;
                  /* find instance of user to assign roles */
                  if (j == instance)
                    {
                        vector <string> roles;
                        for (size_t i = 1; i < numofrole; i++)
                        {
                            string rolename;
                            cout<<"Enter role name"<<i<"\n";
                            cin>>rolename;
                            roles.push_back (rolename);
                        }
                        try
                        {
                            user.assignRoles (roles);
                            fprintf (stdout, "Roles Assigned Successfully\n");
                            return 0;
```

```
                        }
                        catch (exception &e)
                        {
                                fprintf (stderr, "Error: Assigning roles failed\n");
                        }
                        return 0;
                    }
                }
                if (j < instance)
                {
                    fprintf (stdout, "User instance %d is not found",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing user : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of user, & num. of roles to assignroles to  particular user
2       1
Enter role name 1
admin
Roles Assigned Successfully
```

11) How to "delete" a role?

1. The role can be delete .
2. Get the instance of the role to delete role
3. To do that, enumerate the role using CRole::enumRoles.
4. Then the role can be deleted by calling deleteRole function from CRole class as follows.

   CRole::deleteRole.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "computersystem.h"
#include "user.h"
using namespace dsdk;
int
```

```
main (void)
{
        int instance;
        cout <<"Enter the instance of role to delete role\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the user */
                CRole::iterator i = CRole::enumRoles (client);
                /* get the instances of the role and j value willget increment for each role */
                for (int j = 0; i != CRole::iterator::end (); ++i,j++)
                {
                  CRole role = *i;
                  /* find instance of role to delete role */
                  if (j == instance)
                   {
                        if (role.deleteRole () == 0)
                          {
                                fprintf (stdout, "Role deleted successfully\n");
                                return 0;
                          }
                        else
                          {
                                fpritnf (stdout, "Error: Deleting Role\n");
                                return 0;
                          }
                   }
                }

                if (j < instance)
                 {
                     fprintf (stdout, "Role instance %d is not found",instance);
                 }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing role : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of role to delete role
```

```
2
Role deleted Successfully
Enter the instance of role to delete role
14
Role instance 14 is not found
```

12) How to get the speed of the second processor?

To get the list of processors advertised by a MAP.

1. Enumerate the processor using static function CProcessor::enumerateProcessor
2. From the list of processors get the iterate to the second processor.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "processor.h"
using namespace dsdk;
int
main (void)
{
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the processors */
                CProcessor::iterator i = Cprocessor::enumProcessors (client);

                /* get the instances of the processors and j value will get increment for each processor */
                for (int j = 0; i != CProcessor::iterator::end (); ++i,j++)
                {
                        CProcessor proc = *i;
                        /* get the speed of the second processor */
                        if (j == 2)
                        {
                                fprintf (stdout, "Processor Speed : %d\n", proc.getCurrentClockSpeed ());
                                return 0;
                        }
                }
                if (j < 1)
                {
                        fprintf (stdout, "Only one processor is available\n");
                }
        }
        catch (exception &e)
        {
```

```
                fprintf (stdout, "Error accessing processors : %s\n", e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Processor Speed : 3600
```

13)How to change the boot order?

1. Enumerate all the instance of bootconfig using
   CBootConfig::enumBootConfig
2. Get the old boot order list.
3. From that change boot order list according to user options.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "bootconfig.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of bootconfig to change bootorder\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
            /* enumerate the bootconfig */
             CBootConfig::iterator i = CBootConfig::enumBootConfig (client);
             for (int j = 0; i != CBootConfig::iterator::end (); ++i,j++)
                  {
                   /* get the instances of the bootconfig to change bootorder */
                     CBootConfig bc = *i;
                     /* get the existing bootorder */
                     if (j == instance)
                    {
```

```
                    vector <CBootConfig::BootDeviceInfo_T> oldBootOrder = bc.getBootOrder ();
                    /* finding oldbootorder size */
                    size =  oldBootOrder.size ();
                    int newOrder[size];
                    cout <<"Size of oldbootorder is "<<size<<"\n";
                    cout <<"Enter no of boot order to change & new boot order
                     list\n";
                    int toChange;
                    if (toChange > size)
                    {
                    cout <<"The size should not exceed oldbootorder\n";
                    return -1;
                    }
                    cin >>toChange;
                    for (int i = 0; i<toChange; i++)
                    {
                          cin >>newOrder[i];
                    }
                          /* new boot order */
                    vector <CBootConfig::BootDeviceInfo_T> newBootOrder;
                    for (int i = 0; i<size; i++)
                    {
                       newBootOrder.push_back (oldBootOrder[newOrder[i]]);
                    }
                    if (0 == bc.changeBootOrder (newBootOrder))
                    {
                           fprintf (stdout, "Boot Order Changed Successfully\n");
                 }
                    else
                    {
                   fprintf (stderr, "Changing Boot Order Failed\n");
                          return 0;
                    }
                  }
                }
                if (j < instance)
          {
                  fprintf (stdout, "bootconfig instance %d not  available\n",instance);
          }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing bootconfig : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of bootconfig to change bootorder
1
Size of oldbootorder is 3
Enter no of boot order to change & new boot order list
3
2 0 1
Boot Order Changed  Successfully
```

14) How to set speed for fan?

1. Enumerate all the instance of Fan using Cfan::enumFans.
2. Find the instance of the Fan to set speed.
3. Invoke the setDesiredSpeed method of this instance of Fan.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "fan.h"
using namespace dsdk;
int
main (void)
{
        int speed, instance;
        cout<<"Enter the fan instance & speed to be set\n"
        cin>>instance>>speed;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
           /* enumerate the fan */
            CFan::iterator i = CFan::enumFans (client);
            /* get the instances of the fan and j value will get increment for each fan */
            for (int j = 0; i != CFan::iterator::end (); ++i,j++)
            {
                  CFan fan = *i;
           /* get the instance of fan to set speed */
                 if (j == instance)
                  {
                    /* set desired speed */
                    try
                    {
                        uint32 status = fan.SetSpeed (speed);
                          /* if set speed success it will print as success,else throw EFunctionReturnedWithFailure
                                                                                                  execption */
                          if (status == 0)
                          {
```

```
                                fprintf (stdout, "Speed set successfully\n");
                            }
                            else
                            {
                                string retcodestr = fan.
                                        getValueStr_SetSpeed_ReturnCode(status);
                                    throw EFunctionReturnedWithFailure
                                  ("CIM_Fan::SetSpeed", retcodestr, status);
                            }
                        }
                        catch (DSDKError &e)
                        {
                           fprintf (stdout, "Set Speed failed\n");
                        }
                       }
                }
                if (j < instance)
                 {
                        fprintf (stdout, "Fan instance %d not available\n",instance);
                 }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing fan : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the fan instance & speed to be set
1   1000
Set speed successfully
```

15) How to write & Read a data?

1. WriteData & Read data can be done in OpaqueManagementData only
2. Data can be write first and then read the written data
3. To writeData:
   Enumerate the opaquemanagementdata using
   CopaqueManagementData::enumOpaqueManagementData
   Then call the function writeData (offset, &length, data) to write data
4. To readData
   Call the function readData (offset,&length,&data) to read written data.

A sample usage is show below.

```
#include <stdio.h>
```

```
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "opaquemanagementdata.h"
#define MAX_VALUE 50
using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of opaquemanagementdata where data to be write & read\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the opaquemanagementdata */
                COpaqueManagementData::iterator i = COpaqueManagementData::
                                                        enumCOpaqueManagementData (client);
                /* get the instances of the opaquemanagementdata and j value will get increment for each
                                                                opaquemanagementdata */
                for (int j = 0; i != COpaqueManagementData::iterator::end ();++i,j++)
                {
                        CopaqueManagementData omd = *i;
                    if (j == instance)
                     {
                        uint64 Offset; /* offset to read & write */
                        uint64 Length; /* length of data to write */
                        uint8  values[MAX_VALUE];
                        uint32 status;
                        vector<uint8> ReadData;
                        vector<uint8> WriteData;
                        cout<<"Enter offset & no of values to write\n";
                        cin>>Offset>>Length;
                        /* get values to write data */
                        for (int i=0;i<Length;i++)
                        {
                          cout <<"Enter the data"<<i+1<<"to write:\n";
                          cin >>values[i];
                          WriteData.push_back(values[i]);
                        }
                        try
                        {
                                status = omd.writeData (Offset,&Length,WriteData);
                                if (!status}
                                {
                                    fprintf(stdout, "Write Data Success\n");
```

```
                                }
                                else
                                {
                                    fprintf(stdout, "Write Data failed\n");
                                }
                        }
                        catch (...)
                        {
                            fprintf (stdout, "Error accessing writedata\n");
                        }
                        /* read data */
                        try
                        {
                           status = omd.readData(Offset,&Length,&ReadData);
                           if ( !status )
                           {
                                fprintf (stdout, "Read data success\n");
                                for (size_t i=0;i<ReadData.size ();i++)
                                {
                                  fprintf (stdout, "DataRead\n%d",ReadData[i]);
                                }
                           }
                           else
                           {
                                fprintf (stdout, "read data failed\n");
                           }
                        }
                        catch (...)
                        {
                             fprintf (stdout, "Error accessing readdata\n");
                        }
                    }
            if (j < instance)
            {
                fprintf (stdout, "OpaqueManagementData instance %d not found\n",instance);
            }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing opaquemanagementdata : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of opaquemanagementdata where data to be write & read
```

```
1
Enter offset & no of values to write
2506     2
Enter the data 1 to write
25
Enter the data 2 to write
64
Write Data Success
Read Data Success
Data Read
25
64
```

16) How to set attribute value to bios attribute?

1. Attributes can be set to particular bios element only.
2. Get the instance of the bios to set attribute.
3. Get the attributename to set attribute value(attribute name should be same as listed in show command)
4. Invoke the setattrubute method of this instance of bios.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "biosmanagement.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of bios to setattribute\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the bios element */
                CBIOSElement::iterator i = CBIOSElement::enumCBIOSElement (client);
                /* get the instances of the bios elemene and j value will get increment for each bios element */
                for (int j = 0; i != CBIOSElement::iterator::end (); ++i,j++)
                {
                    CBIOSElement be = *i;
                   if (j == instance)
                    {
```

```
                    size_t count;
                    int noofvalues;
                    string attrName;
                    vector <string> values;
                    cout <<"Enter the attribute name to set value & no of values to set\n";
                    cin >>attrName;
                    cin >>noofvalues;
                    for (size_t i = 0; i < noofvalues; i++)
                    {
                        string attrValue;
                        cout <<"Enter the value"<<i<<"\n";
                        cin >>attrValue;
                        values.push_back (attrValue);
                    }

                    /* get the attribute to set */
                    vector <CBIOSAttribute> ba =  be.getAttributes ();
                    for (count = 0; count < ba.size(); count++)
                    {
                         if(ba[count].getName()== attrName)
                            { break; }
                    }
                    if (count == ba.size())
                    {
                         fprintf (stderr, "Invalid: AttributeName\n");
                         return -1;
                    }

                    try
                    {
                        ba[count].setAttribute(values);
                        fprintf (stdout, "Sucess: SetAttributes \n");
                    }
                    catch (exception &e)
                    {
                        fprintf (stderr, "Error: Set Attribute failed\n");
                        return -1;
                    }
                }
            }
            if (j < instance)
            {
                fprintf (stdout,"bios instance %d not found\n",instance);
        }
    }
    catch (exception &e)
    {
            fprintf (stdout, "Error accessing bios : %s\n",e.what());
    }
```

```
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of bios to setattribute
1
Enter the attribute name to set value & no of values to set
DMTF:IDEController
1
Enter the value 1
Enable
Success: SetAttributes
```

17) How restore BIOS to default setting?

1. It will restore the bios with default values.
2. To do that,get the instance of the bios to restore defaults.
3. Invoke the restoreDefaults method of this instance to restore default values to bios.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "biosmanagement.h"
using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of bios to restore defaults\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the bios element */
                CBIOSElement::iterator i = CBIOSElement::enumCBIOSElement (client);
                /* get the instances of the bios elemene and j value  will get increment for each bios element */
                for (int j = 0; i != CBIOSElement::iterator::end ();++i,j++)
                {
                    CBIOSElement be = *i;
                    if (j == instance)
                    {
```

```
                        try
                          {
                            be.restoreDefaults();
                            fprintf (stdout, "Restored default BIOS successfully\n");
                            return;
                          }
                        catch (exception &e)
                          {
                             fprintf (stderr, "Error: RestoreDefaults failed\n");
                          }
                          return -1;
                        }
                    }
                    if (j < instance)
                    {
                         fprintf (stdout,"bios instance %d not found\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing bios : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of bios to restore defaults
1
Restored default BIOS successfully
```

18) How to do a Battery “enable” ?

1. Enumerate all the instance of Battery using CBattery::enumBattery
2. Find the instance of the battery to be on.
3. Invoke the enable method of this instance of battery.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "battery.h"
using namespace dsdk;

int
main (void)
{
```

```
        string deviceID;
        cout <<"Enter the device id of the battery to be on\n";
        cin >>deviceID;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the battery */
                CBattery::iterator i = CBattery:: enumBattery (client);
                /* get the instances of the battery */
                for (; i != CBattery::iterator::end (); ++i)
                {
                        CBattery bat = *i;
                        /* find the battery to be on */
                        if (bat.deviceID () == deviceID)
                        {
                                if (bat.enable () == 0)
                                {
                                    fprintf (stdout, "Battery Enabled successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Enable Battery\n");
                                    return 0;
                                }
                        }
                }
                else
                {
                     fprintf (stdout, "Battery device id %s is not available\n",deviceID);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing battery : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the device ID of the battery to be on
battery1
Battery Enabled successfully

Enter the device ID of the battery to be on
```

```
battery6
Battery device id 6 is not available
```

19) How to do a Battery “disable” ?

1. Enumerate all the instance of Battery using CBattery::enumBattery
2. Find the instance of the battery to be off.
3. Invoke the disable method of this instance of battery.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "battery.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of the battery to be off\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
                /* enumerate the battery */
                CBattery::iterator i = CBattery::enumBattery (client);
                /* get the instances of the battery and j value will get increment for each battery */
                for (int j = 0; i != CBattery::iterator::end (); ++i,j++)
                {
                        CBattery bat = *i;
                        /* find the battery to be off */
                        if (j == instance)
                        {
                                if (bat.disable () == 0)
                                {
                                    fprintf (stdout, "Battery Disabled successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Disable Battery\n");
                                    return 0;
                                }
                        }
```

```
                }
                if (j < instance)
                {
                        fprintf (stdout, "Battery instance %d is not available\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing battery : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of the battery to be off
1
Battery Disabled successfully

Enter the instance of the battery to be off
5
Battery instance 5 is not available
```

20)How to do a Battery "test" ?

1. Enumerate all the instance of Battery using CBattery::enumBattery
2. Find the instance of the battery to perform a recalculation of charge thresholds.
3. Invoke the test method of this instance of battery.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "battery.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of the battery to perform a recalculation of charge thresholds\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
```

```
        try
        {
                /* enumerate the battery */
                CBattery::iterator i = CBattery::enumBattery (client);
                /* get the instances of the battery and j value will  get increment for each battery */
                for (int j = 0; i != CBattery::iterator::end (); ++i,j++)
                {
                        CBattery bat = *i;

                        /* find the battery to perform recalculation of charge thresholds */
                        if (j == instance)
                        {
                                if (bat.test () == 0)
                                {
                                    fprintf (stdout, "Battery Test successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Test Battery\n");
                                    return 0;
                                }
                        }
                }
                if (j < instance)
                {
                     fprintf (stdout, "Battery instance %d is not available\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing battery : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of the battery to perform a recalculation of charge thresholds
1
Battery Test successfully

Enter the instance of the battery to perform a recalculation of charge thresholds
6
Battery instance 6 is not available
```

21) How to do a Battery “recharge” ?

1. Enumerate all the instance of Battery using CBattery::enumBattery

2. Find the instance of the battery to do recharge operation.
3. Invoke the reset method of this instance of battery.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "battery.h"

using namespace dsdk;

int
main (void)
{
        int instance;
        cout <<"Enter the instance of the battery to recharge\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);

        try
        {
                /* enumerate the battery */
                CBattery::iterator i = CBattery:: enumBattery (client);
                /* get the instances of the battery and j value will get increment for each battery */
                for (int j = 0; i != CBattery::iterator::end (); ++i,j++)
                {
                        CBattery bat = *i;

                        /* find the battery to recharge */
                        if (j == instance)
                        {
                                if (bat.reset () == 0)
                                {
                                    fprintf (stdout, "Battery recharge successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Recharge Battery\n");
                                    return 0;
                                }
                        }
                }
                if (j < instance)
                {
```

```
                        fprintf (stdout, "Battery instance %d is not available\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing battery : %s\n",
            e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of the battery to recharge
1
Battery recharge successfully

Enter the instance of the battery to recharge
5
Battery instance 5 is not available
```

22) How to “activate” a session in Text Redirection ?

1. Enumerate all the instance of TextRedirection using CTextRedirection::enumTextRedirections
2. Find the instance of the TextRedirection to activate a session.
3. Invoke the activate method of this instance of textredirection.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "textredirection.h"

using namespace dsdk;
int
main (void)
{
        int instance;
        cout <<"Enter the instance of the textredirection to activate a session\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
```

```
        try
        {
            /* enumerate textredirection */
            CTextRedirection::iterator i = CTextRedirection::enumTextRedirections (client);
            /* get the instances of textredirection and j value will get increment for each instance */
            for (int j = 0; i != CTextRedirection::iterator::end (); ++i,j++)
            {
                        CTextRedirection tr = *i;
                        /* find the textredirection instance to activate */
                        if (j == instance)
                        {
                                if (tr.activate () == 0)
                                {
                                    fprintf (stdout, "Session activated successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Activating Session\n");
                                    return 0;
                                }
                        }
                }
                if (j < instance)
                {
                     fprintf (stdout, "TextRedirection instance %d is not available\n",instance);
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing textredirection : %s\n",e.what());
        }
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the instance of the textredirection to activate a session
1
Session activated successfully

Enter the instance of the textredirection to activate a session
5
TextRedirection instance 5 is not available
```

23) How to “deactivate” a session in Text Redirection ?

1. Enumerate all the instance of TextRedirection using

CTextRedirection::enumTextRedirections

2. Find the name of the TextRedirection to deactivate a session.
3. Invoke the deactivate method of this instance of textredirection.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "textredirection.h"

using namespace dsdk;

int
main (void)
{
        string name;
        cout <<"Enter the name of the textredirection to deactivate a session\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
            /* enumerate textredirection */
            CTextRedirection::iterator i = CTextRedirection::enumTextRedirections (client);
            /* get the instances of textredirection and j value will get increment for each instance */
            for (; i != CTextRedirection::iterator::end (); ++i)
            {
                        CTextRedirection tr = *i;
                        /* find the textredirection instance to deactivate */
                        if (tr.getName () == name)
                        {
                                if (tr.deactivate () == 0)
                                {
                                    fprintf (stdout, "Session deactivated successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Deactivating Session\n");
                                    return 0;
                                }
                        }
                }
                else
                {
                    fprintf (stdout, "TextRedirection name %sis not available\n",name);
```

```
                }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing textredirection : %s\n",e.what());
        }

        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the nmae of the textredirection to deactivate a session
textredirectsap_ssh
Session deactivated successfully

Enter the name of the textredirection to deactivate a session
textredirectsap_ssh5
TextRedirection name textredirectsap_ssh5 is not available
```

24) How to “disable” a session in Text Redirection ?

1. Enumerate all the instance of TextRedirection using CTextRedirection::enumTextRedirections
2. Find the instance of the TextRedirection to disable a session.
3. Invoke the disable method of this instance of textredirection.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "textredirection.h"
using namespace dsdk;
int
main (void)
{
        int instance;
        cout <<"Enter the instance of the textredirection to disable a session\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
            /* enumerate textredirection */
```

```
        CTextRedirection::iterator i = CTextRedirection::enumTextRedirections (client);
        /* get the instances of textredirection and j value will get increment for each instance */
        for (int j = 0; i != CTextRedirection::iterator::end (); ++i,j++)
        {
                CTextRedirection tr = *i;
                /* find the textredirection instance to disable */
                if (j == instance)
                {
                        if (tr.disable () == 0)
                        {
                            fprintf (stdout, "Session disabled successfully\n");
                            return 0;
                        }
                        else
                        {
                            fpritnf (stdout, "Error: Disabling Session\n");
                            return 0;
                        }
                }
            }
            if (j < instance)
            {
                fprintf (stdout, "TextRedirection instance %d is not available\n",instance);
            }
    }
    catch (exception &e)
    {
            fprintf (stdout, "Error accessing textredirection : %s\n",e.what());
    }

    delete client;
    return 0;
}
```

A sample input & output is below

```
Enter the instance of the textredirection to disable a session
1
Session disabled successfully

Enter the instance of the textredirection to disable a session
5
TextRedirection instance 5 is not available
```

25) How to “start” a redirection session in Text Redirection ?

1. Enumerate all the instance of TextRedirection using CTextRedirection::enumTextRedirections
2. Find the name of the TextRedirection to start a redirection session.
3. Invoke the start method of this instance of textredirection.

A sample usage is show below.

```
#include <stdio.h>
#include "subject.h"
#include "cimmap.h"
#include "subject.h"
#include "textredirection.h"
using namespace dsdk;
int
main (void)
{
        string name;
        cout <<"Enter the name of the textredirection to start a redirection session\n";
        cin >>instance;
        CCIMMAP cm ("192.168.0.20", "623");
        CSubject subject ("admin", "admin","digest");
        IClient* client = cm.connect (subject);
        try
        {
            /* enumerate textredirection */
            CTextRedirection::iterator i = CTextRedirection::enumTextRedirections (client);
            /* get the instances of textredirection and j value will get increment for each instance */
            for (; i != CTextRedirection::iterator::end (); ++i)
            {
                    CTextRedirection tr = *i;
                    /* find the textredirection instance to start a session */
                        if (tr.getName () == name)
                        {
                                if (tr.start () == 0)
                                {
                                    fprintf (stdout, "Session Started successfully\n");
                                    return 0;
                                }
                                else
                                {
                                    fpritnf (stdout, "Error: Starting redirection Session\n");
                                    return 0;
                                }
                        }
            }
            else
            {
                    fprintf (stdout, "TextRedirection name %s is not available\n",name);
            }
        }
        catch (exception &e)
        {
                fprintf (stdout, "Error accessing textredirection : %s\n",e.what());
        }
```

```
        delete client;
        return 0;
}
```

A sample input & output is below

```
Enter the name of the textredirection to start a redirection session
textredirection_ssh
Session Started successfully

Enter the name of the textredirection to start a redirection session
textredirection_ssh2
TextRedirection name textredirection_ssh2 is not available
```

4 Low Level API-C++

The Low Level C++ API is defined to match the CMPI interface. The following is the list of Low level API Classes.

- IClient
- CCIMObjectPath
- CCIMInstance
- CCIMEnumerator
- CCIMValue
- CCIMData
- CCIMString
- CCIMArray
- CCIMArgument

Note: When using these low level C++ API's below rule need to be followed on when to set auto_release true/false.

When a high level object is created using low level CMPI object that is not a member/property of another object or not a element in a array, the auto_release should be set to true(default).

Example:

```
CMPIObjectPath  cmpi_op;//a low level object
CCIMObjectPath op = CCIMObjectPath::toCCIMObjectPath (cmpi_op); //create a high level object, using the low level object cmpi_op that is not member of other object or element of array.
```

When a high level object is created using low level CMPI object that is a member/property of another object or a element in a array the auto_release should be set to false(as this object will be released when the a object/array containing this object is released).

Ex.

```
CMPIInstance  cmpi_inst; //a low level object
CCIMObjectPath op = CCIMObjectPath::toCCIMObjectPath (cmpi inst.getObjectPath(), false); //create a high level object using the low level object cmpi_inst.getObjectPath () that is member of other object

CMPIArray cmpi_op_array; //a low level array
CCIMObjectPath op = CCIMObjectPath::toCCIMObjectPath (cmpi_op_array[i], false); //create a high level object, using the low level object cmpi_op_array[i] that is a element in a array.
```

4.1 CCIMObjectPath

A Class representing a CIM Object path. This class provides a C++ wrapper and mostly delegates the functions to the low level C API.

Member Functions

- setNameSpace
- getNameSpace
- setHostName
- getHostName
- setClassName
- getClassName
- addKey
- getKey
- getKeyCount
- getKeyAt
- setNameSpaceFromObjectPath
- setHostAndNameSpaceFromObjectPath
- getClassQualifier
- getPropertyQualifier
- getMethodQualifier
- getParameterQualifier
- toString

Static Member Functions:

- toCCIMObjectPath
- create
- instanceToObjectPath

constructor description:

- CCIMObjectPath
  Description: Construct the object from the low level object.
  Syntax: CCIMObjectPath (const string& ns, const string& object_name)
  Parameter:
    - *ns* The namespace relative to the current namespace.
    - *object_name* The name of the CIM element.

Member Function Description:

- toCCIMObjectPath
  Description: Convert the low level object path to CCIMObjectPath. The low level object path should not be released or assigned to another object after this call. We could have cloned this low level object path but this would incur an unnecessary memory allocation since most of the use cases will be just a straight conversion from low level object path.
  Syntax: CCIMObjectPath toCCIMObjectPath (CMPIObjectPath* path,bool _auto_release = true);
  Parameters:
  - *path* namespace to objectpath

- **create**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMObjectPath create (CMPIValue* val, bool auto_release)
  Parameters:
  - *val* low level value

- **instanceToObjectPath**
  Descriptions: A static function to convert a low level instance to an object path
  Syntax: CCIMObjectPath instanceToObjectPath (CMPIInstance* inst, bool auto_release = false);
  Parameter:
  - *inst* low level instance

- **setNameSpace**
  Description: Set/replace the nameSpace component.
  Syntax: void setNameSpace (const string& ns) ;
  Parameters:
  - *ns* The nameSpace string
  Returns: none

- **getNameSpace**
  Description: Get the nameSpace component.
  Syntax: string getNameSpace (void) const ;
  Return: The namespace component.

- **setHostName**
  Description: Set/replace the host name component.
  Syntax: void setHostName (const string& hn) ;
  Parameter:

- *hn* The host name string

Return: none

- **getHostName**

  Description: Get the host name component.

  Syntax: string getHostName (void) const ;

  Return: The host name component.

- **setClassName**

  Description: Set/replace the class name component.

  Syntax: void setClassName (const string& cn);

  Return: none

- **getClassName**

  Description: Get the class name component.

  Syntax: string getClassName (void) const ;

  Return: none

- **addKey**

  Description: Adds/replaces a named key property.

  Syntax: void addKey (const string& name, const CCIMValue& value) ;

  Parameters:

  - *name* Key property name.
  - *value* Address of value structure.

  Return: none

- **getKey**

  Description: Gets a named key property value.

  Syntax: CCIMData getKey (const string& name) const ;

  Parameters:

  - *name* Key property name.

  Return Entry value.

- **getKeyCount**

  Description: Gets the number of key properties contained in this object path.

  Syntax: unsigned int getKeyCount (void) const ;

  Return The number of properties.

- **getKeyAt**

  Description: Gets a key property value defined by its index.

  Syntax: CCIMData getKeyAt (unsigned int index, string* name) const ;

  Parameters:

- *index* Position in the internal Data array.
- *name* Output: Returned property name (suppressed when NULL).

Return Data value.

- **setNameSpaceFromObjectPath**

Description: Set/replace hostname, nameSpace and classname components from src

Syntax: void setHostAndNameSpaceFromObjectPath(const qCCIMObjectPath&src) ;

Parameters:

- *src* Source input.

Return: none

- **setHostAndNameSpaceFromObjectPath**

Description: Set/replace hostname, nameSpace and classname components from src

Syntax: void setHostAndNameSpaceFromObjectPath (const CCIMObjectPath& src) throw (ECIMError);

Parameters:

- *src* Source input.

Return: none

- **getClassQualifier**

Description: Get class qualifier value.

Syntax: CCIMData getClassQualifier (const string& qual_name) const ;

Parameter:

- *qual_name* Qualifier name.

Return: Qualifier value.

- **getPropertyQualifier**

Description: Get property qualifier value.

Syntax: CCIMData getPropertyQualifier (const string& prop_name,const string& qual_name);

Parameter

- *prop_name* Property name.
- *qual_name* Qualifier name.

Return Qualifier value.

- **getMethodQualifier**

Description: Get method qualifier value.

Syntax: CCIMData getMethodQualifier (const string& method_name, const string& qual_name) const ;

Parameters:

- *method_name* Method name.
- *qual_name* Qualifier name.

Return Qualifier value.

- **getParameterQualifier**

  Description: Get method parameter quailifier value.

  Syntax: CCIMData getParameterQualifier (const string& method_name,
  const string& param_name,
  const string& qual_name) const ;

  Parameter:
  - *method_name* Method name.
  - *param_name* Parameter name.
  - *qual_name* Qualifier name.

  Return Qualifier value.

- **toString**

  Description: Generates a well formed string representation of this ObjectPath.

  Syntax: string toString (void) const throw (ECIMError);

  Return: String representation.

4.2 CCIMInstance

A Class representing an CIM Instance. This class provides a C++ wrapper and mostly delegates the functions to the low level C API.

Member Functions

- clone
- getProperty
- getPropertyCount
- getPropertyAt
- setProperty
- setProperty
- getQualifier
- getQualifierCount
- getQualifierAt
- getPropertyQualifier
- getPropertyQualifierCount
- getPropertyQualifierAt
- getLowLevelObject
- getObjectPath
- setObjectPath
- makeInstanceObjectPath

Static Member Functions:

- toCCIMInstance
- create
- toCMPIValue

Constructor Description:

- **CCIMInstance**
  Description: Construct a CIM Instance from the object path
  Syntax: CCIMInstance (const CCIMObjectPath& op);
  Parameter:
  - *op* objectpath

MemberFunction Description:

- **toCCIMInstance**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMInstance create (CMPIValue* val, bool auto_release
  Parameter:
  - *val* low level CMPIValue

- **create**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMInstance create (CMPIValue* val, bool auto_release)
  Parameter:
  - *val* low level CMPIValue

- **toCMPIValue**
  Description: Converts val to a low level CMPIValue object.
  Syntax: static CMPIValue toCMPIValue (CCIMInstance val)
  Parameter:
  - *val* low level CMPIValue
- **clone**
  Description: Clone this object
  Syntax: CCIMInstance* clone (void);

- **getProperty**
  Description: Gets a named property value.
  Syntax: CCIMData getProperty (const string& name) const ;
  Parameter:
  - *name* The name string
  Return The named property value

- **getPropertyCount**
  Description: Gets the number of properties contained in this instance.
  Syntax: unsigned int getPropertyCount (void) const;
  Return: The number of properties contained in this instance.

- **getPropertyAt**
  Description: Gets a property value defined by its index.
  Syntax: CCIMData getPropertyAt (unsigned int index, string* name) const;
  Return property value defined by its index.

- **setProperty**
  Description: Adds / replace a named property.
  Syntax: void setProperty (const string& name, const CCIMValue& value);
  Parameters:
    - *name* Key property name.
    - *value* Address of value structure.

- **setPropertyFilter**
  Description: Directs the low level layer to ignore any setProperty operations for this instance for any properties not in this list.
  Syntax: void setPropertyFilter (const vector<string>& props,
  const vector<string>& keys);
  Parameters:
    - *props* The property string array
    - *keys* key value string array

- **getQualifier**
  Description: Gets the qualifier.
  Syntax: CCIMData getQualifier (const string& name) const;
  Parameter:
    - *name* The name string to get qualifier

  Return The qualifier
- **getQualifierCount**
  Descripton: Gets the qualifier count.
  Syntax: unsigned int getQualifierCount (void) const;
  Return: The qualifier count

- **getQualifierAt**
  Description: Gets the qualifier at the index.
  Syntax: CCIMData getQualifierAt (unsigned int index, string* name) const;
  Parameters:

- *index* Position in the internal Data array.
- *name* Output Returned property name (suppressed when NULL).

Return: the qualifier at the index.

- **getPropertyQualifier**

Description: Gets the property qualifier.

Syntax: CCIMData getPropertyQualifier (const string& pname, const string& qname);

Parameters:

- *pname* property name string
- *qname* qualifier name string

Return: The property qualifier.

- **getPropertyQualifierCount**

Description: Gets the property qualifier count.

Syntax: unsigned int getPropertyQualifierCount (const string& pname) const;

Parameter:

- *pname* property name string

Return: The property qualifier count

- **getPropertyQualifierAt**

Description: Gets the property qualifier at the specified index.

Syntax: CCIMData getPropertyQualifierAt (const string& pname, unsigned int index,string* qname);

Parameters:

- *pname* property name string
- *index* Position in the internal Data array.
- *qname* qualifier name string

Return: The property qualifier

- **getLowLevelObject**

Description: Gets the low level instance.

Syntax: CMPIInstance* getLowLevelObject (void) const;

Return; The lw level instance

- **getObjectPath**

Description: Generates an object path out of the namespace, classname and key properties of this instance.

Syntax: CCIMObjectPath getObjectPath (void) const;

- **setObjectPath**

Description: Sets the object path corresponding to this instance.

Syntax: void setObjectPath (const CCIMObjectPath& op);
Parameter:

- *op* object path to set

- **makeInstanceObjectPath**
  Description: Construct the object path using class/key properties table
  Syntax: friend void makeInstanceObjectPath (CCIMInstance* inst, const string& classname);
  Parameter:
  - *inst* instance to construct objectpath
  - *classname* class to make instance objectpath

4.3 IClient:

A client interface.

Member Functions

- enumInstanceNames
- enumInstances
- getInstance
- createInstance
- setInstance
- deleteInstance
- associators
- associatorNames
- references
- referenceNames
- invokeMethod
- setProperty
- getProperty
- subscribeEvent
- unsubscribeEvent
- renewSubscription
- getHostName
- getPort
- getUser
- getPassword
- setTimeout
- getTimeout

- **enumInstanceNames**

Description: Enumerate instance names of the class (and subclasses) defined by op op.This function takes an extra list of key properties. This function was added to support implementations which do not support EnumerateMode=EnumerateEPR. Instead this function (if global variable g_enum_names_use_keyprops is enabled) uses the enumerateInstances with the query for key properties and build the object paths from the instances.

syntax: virtual CCIMObjectPath::iterator enumInstanceNames(const CCIMObjectPath& op,const vector<string>& key_props) = 0;

parameters:

- *op* Object path containing the namespace and classname.
- key_props An array of property names.

Return The iterator to the object paths.

- **enumInstances**

Description: Enumerate instances of the class (and subclasses) defined by objectpath

Syntax: virtual CCIMInstance::iterator enumInstances (const CCIMObjectPath& op, CMPIFlagsflags,const vector<string>& props = vector <string>());

Parameters:

- *op* Object path containing namespace and classname components.
- *flags* Reserved
- *props* An array of property names.

Return The iterator to the instances.

- **getInstance**

Description: Get instance using objectpath as reference.

Syntax: virtual CCIMInstance getInstance (const CCIMObjectPath& op,flags,const vector<string>& props = vector <string>());

Parameter:

- *op* Object Path containing namespace, classname and key components.
- *flags* Reserved.
- *props* An array of property names.

Return The instance.

- **createInstance**

Description: Create instance from objectpath inst using objectpath as reference.

Syntax: virtual CCIMObjectPath createInstance (const CCIMObjectPath& op,const CCIMInstance& inst);

Parameter:

- *op* Object path containing namespace, classname and key components.
- *inst* Complete instance.

Return The assigned instance reference.

- **setInstance**

Description: Replace an existing instance from objectpath inst using objectpath as reference.

Syntax: virtual void setInstance (const CCIMObjectPath& op,const CCIMInstance& inst,CMPIFlags flags, const vector<string>& props);

Parameter:

- *op* ObjectPath containing the namespace, classname and key components.
- *inst* Complete instance.
- *flags* Reserved
- *props* An array of property names.

Return None.

- **deleteInstance**

Description: Delete an existing instance using objectpath as reference.

Syntax: virtual void deleteInstance (const CCIMObjectPath& op);

Parameters:

- *op* Object path containing namespace, classname and key components.

Return none

- **associators**

Description: Enumerate instances associated with the instance defined by the objectpath.

Syntax: virtual CCIMInstance::iterator associators (const CCIMObjectPath& op,const string& assoc_class, const string& result_class, const string& role, const string&result_role, CMPIFlags flags, const vector<string>& props = vector <string>());

Parameters:

- *op* Source object path containing namespace, classname and key components.
- *assoc_class* If not empty must be a valid association class name. It acts as a filter on the returned set of objects by mandating that each returned object must be associated to the source object via an Instance of this class or one of its subclasses.
- *result_class* If not null, MUST be a valid class name. It acts as filter on the returned set of objects by mandating that each returned object

must be either an Instance of this class (or its subclass).

- *role* If not empty, MUST be a valid property name.It acts as a filter on the the returned set of objects by mandating that each returned object MUST be associated to the source object via an Association in which the source object plays the specified role (i.e. the name of the property in the association class that refers to the source object must match the value of this parameter).
- *result_role* If not empty, MUST be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object MUST match the value of this parameter).
- *flags* Reserved.
- *props* If not empty, the members of the array define one or more Property names. Each returned Object MUST NOT include elements for any Properties missing from this list.

Return Iterator to the instances.

- **associatorNames**

Desciption: Enumerate ObjectPaths associated with the Instance defined by objectpath.

Syntax: virtual CCIMObjectPath::iterator associatorNames (const CCIMObjectPath& op, const string& assoc_class, const string& result_class, const string& role, const string&result_role);

Parameters:

- *op* Source ObjectPath containing nameSpace, classname and key components.
- *assoc_class* If not empty, MUST be a valid Association Class name.It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Instance of this Class or one of its subclasses.
- *result_class* If not empty, MUST be a valid Class name.It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be either an Instance of this Class (or one of its subclasses).
- *role* If not NULL, MUST be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object MUST match the value of this parameter).
- *result_role* If not NULL, MUST be a valid Property name. It acts as a filter on

the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Association in which the returned Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the returned Object MUST match the value of this parameter).

Return: An iterator to the object paths

- **references**

  Descriptons: Enumerates the association instances that refer to the instance defined by objectpath.

  Syntax: virtual CCIMInstance::iterator references (const CCIMObjectPath& op, const string&result_class, const string& role, CMPIFlags flags, const vector<string>& props = vector <string>());

  Parameters:

  - *op* Source ObjectPath containing nameSpace, classname and key components.
  - *result_class* If not NULL, MUST be a valid Class name.It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be either an Instance of this Class (or one of its subclasses).
  - *role* If not NULL, MUST be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object MUST match the value of this parameter).
  - *flags* Reserved.
  - *props* If not empty, the members of the array define one or more Property names. Each returned Object MUST NOT include elements for any Properties missing from this list

  Return An iterator to the instances.

- **referenceNames**

  Description: Enumerates the association object paths that refer to the instance defined by objectpath.

  Syntax: virtual CCIMObjectPath::iterator referenceNames (const CCIMObjectPath& op, const string& result_class, const string& role) = 0;

  Parameters:

  - *op* Source ObjectPath containing nameSpace, classname and key components.

- *result_class* If not NULL, MUST be a valid Class name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be either an Instance of this Class (or one of its subclasses).
- *role* If not NULL, MUST be a valid Property name. It acts as a filter on the returned set of Objects by mandating that each returned Object MUST be associated to the source Object via an Association in which the source Object plays the specified role (i.e. the name of the Property in the Association Class that refers to the source Object MUST match the value of this parameter).

Return An iterator to the object paths.

- **invokeMethod**

Description: Invoke a named, extrinsic method of an Instance defined by objectpath parameter.

Syntax: virtual CCIMData invokeMethod (const CCIMObjectPath& op, const string&method, const CCIMArgument& in, CCIMArgument* out) ;

Parameters:

- *op* ObjectPath containing nameSpace, classname and key components.
- *method* Method name
- *in* Input parameters.
- *out* Output parameters.

Return Method return value.

- **setProperty**

Description: Set the named property value of an Instance defined by the objectpath parameter.

Syntax:
```
virtual void setProperty (const CCIMObjectPath& op,
                          const string&          name,
                          const CCIMValue&       value) = 0;
```

Parameters:

- *op* ObjectPath containing nameSpace, classname and key components.
- *name* Property name
- *value* Value.

Return None.

- **getProperty**

Description: Get the named property value of an Instance defined by objectpath parameter.

Syntax: virtual CCIMValue getProperty (const CCIMObjectPath& op,

const string& name) = 0;

Parameters:

- *op* ObjectPath containing nameSpace, classname and key components.
- *name* Property name

Return Property value.

- **subscribeEvent**

Description: Subscribe to an event.

Syntax:
```
virtual string subscribeEvent (const string&  delivery_uri, int mode,
                               float heartbeat_interval,
                               float expiration_timeout,
                               const string&  dialect,
                               const string&  filter,
                               const string&  resource_uri =
                               "http://schemas.dmtf.org/wbem/wscim/1/*") = 0;
```

Parameters:

- *delivery_uri* The URI to which the event needs to be delivered.
- *mode* MODE_PUSH, MODE_PUSH_ACK, MODE_PULL
- *heartbeat_interval* The interval in which heartbeat events are sent.
- *expiration_timeout* The timeout by which the subscription expires.
- *filter* Event filter.
- *resource_uri*
- *dialect*

Return The UUID of the subscription.

- **unsubscribeEvent**

Description: Un subscribe a previously subscribed event.

Syntax:
```
virtual void unsubscribeEvent (const string& uuid);
```

Parameters:

- *uuid* The UUID of the subscription.

Return none

- renewSubscription

Description: Renew an existing subscription.

Syntax:
```
virtual void renewSubscription (const string& uuid);
```

Parameters:

- *uuid* The UUID of the subscription.

Return none

- **getHostName**

Description: Gets the Host name for this client

Syntax: virtual string getHostName (void);
Return The hostname

- **getPort**
  Description: Gets the port for this client
  Syntax: virtual int getPort (void);
  Return The Port

- **getUser**
  Description: Gets the user name for this client
  Syntax: virtual string getUser(void);
  Return The Username

- **getPassword**
  Description: Gets the password for this client
  Syntax: virtual string getPassword (void);
  Return The Password

- **setTimeout**
  Description: Sets the time out for this client
  Syntax: virtual void setTimeout (unsigned long timeout);
  Parameters:
    - *timeout* Timeout value to set.
  Return none

- **getTimeout**
  Description: Gets the time out for this client
  Syntax: virtual unsigned long getTimeout (void);
  Return: Timeout period

4.4 ECIMInvalidData:

An exception thrown when accessing an invalid data.

Member Functions:

- getLowLevelType
- isGoodValue
- isNullValue
- isKeyValue
- isNotFound
- isBadValue

- isValid
- getValue

Static Member Functions:

- toCCIMData

Constructor Description:

- **CCIMData**
  Description: Construct a CIM Datafrom the CMPI Data
  Syntax: CCIMData (const CMPIData& data)
  Parameter:
    - *data* CMPI Data

Member Function Description:

- **toCCIMData**
  Description: Convert from the low level object.
  Syntax: static CCIMData toCCIMData (CMPIData data, bool auto_release = true);
  Parameters:
    - *data* CMPI Data to convert

- **isGoodValue**
  Description: Checks if this data is good.
  Syntax: inline bool isGoodValue (void) const
  Returns The state as true if good else false

- **isNullValue**
  Description: Checks if this data is null
  Syntax: inline bool isNullValue (void) const
  Returns The state as true if null else false

- **isKeyValue**
  Description: Checks if this is key value
  Syntax: inline bool isKeyValue (void) const
  Returns The state is true if it is key value else false

- **isNotFound**
  Description: Checks if the state is not found
  syntax: inline bool isNotFound (void) const
  retuns: True if found else False

- **isBadValue**

Description: Checks if this is a bad value.
Syntax: inline bool isBadValue (void) const
Returns True if bad value, else false

- **isValid**
  Description: Checks if this is a valid value.
  Syntax: inline bool isValid (void) const
  Returns True if it is valid, else false

- **getValue**
  Description: Gets the value of this data
  Syntax: inline CCIMValue getValue (void) const
  Returns The value

4.5 EInvalidValueObject

An exception thrown when accessing an invalid value object.

Member Functions:

- getLowLevelType
- getLowLevelValue
- isValid
- invalidate
- clone

Static Member Functions:

- toCCIMValue

Constructor Description:

- **EInvalidValueObject**
  Description: Construct an invalid object from CMPI type
  Syntax: EInvalidValueObject (const CMPIType& type)
  Parameter:
  - *type* CMPI type

Member Function Description:

- **toCCIMValue**
  Description: Builds CCIMValue object from the low level value and type objects. The responsibility of freeing the low level objects is handed over to the object that is created and should not be done by the caller.
  Syntax: static CCIMValue toCCIMValue (const CMPIValue& value, CMPIType type,

bool auto_release = false);

Parameter:

- *value* low level value object
- *type* low level type objects

- **getLowLevelType**

  Description: Returns the underlying value type
  Syntax: CMPIType getLowLevelType (void) const
  Returns low level type object

- **getLowLevelValue**

  Description: Returns the underlying value
  Syntax: CMPIValue* getLowLevelValue (void) const
  Returns low level value object

- **isValid**

  Description: Checks if this object is valid
  Syntax: bool isValid (void) const
  Return true if valid else false

- **invalidate**

  Description: Invalidates this object (and any copies)
  Syntax: void invalidate (void)
  Returns none

- **clone**

  Description: Clone this object
  Syntax: CCIMValue clone (void) const;
  Returns none

4.6 CCIMEnumeration

A class representing CIM Enumeration.

Member Functions:

- setNameSpace
- getClassName
- setClassName
- clone
- getLowLevelObject
- getNext
- hasNext

Static Member Functions:

- toCCIMEnumeration
- create
- toCMPIValue

Constructor Description:

- **CCIMEnumeration**

  Description: Constructor taking CMPIEnumeration as argument.

  Syntax: CCIMEnumeration (const CCIMEnumeration& en);

  Parameters:

  - *en* Low Level Enumeration

Member Function Description:

- **toCCIMEnumeration**

  Description: Convert the low level Enumeration to CCIMEnumeration. The low level enumeration should not be released or assigned to another object after this call. We could have cloned this low level argument but this would incur an unneccessary memory allocation since most of the use cases will be just a straight conversion from low level argument.

  Syntax: static CCIMEnumeration toCCIMEnumeration (CMPIEnumeration* en, bool auto_release = true);

  Parameters:

  - *en* Low Level Enumeration

- **create**

  Description: Creates this object from a low level CMPIValue.

  Syntax: static CCIMEnumeration create (CMPIValue* val, bool auto_release) ;

  Parameters:

  - *val* Low Level CMPIValue

- **toCMPIValue**

  Description: Converts Low Level CMPIValue to a low level CMPIValue object.

  Syntax: static CMPIValue toCMPIValue (CCIMEnumeration val);

  Parameters:

  - *val* Low Level CMPIValue

  Returns CMPIValue value object

- **setNameSpace**

  Description: Sets the name space corresponding to this instance.

Syntax: void setNameSpace (const string& ns);
Parameters:

- *ns* namespace to set

- **getClassName**
  Description: Gets the class name corresponding to this instance.
  Syntax: string getClassName (void) const;
  Returns: The class name

- **setClassName**
  Description: Sets the class name corresponding to this instance.
  Syntax: void setClassName (const string& classname);
  Parameters:
  - *classname* Class Name to set

- **clone**
  Description: Clone this object
  Syntax: CCIMEnumeration* clone (void) const;
  Returns none

- **getLowLevelObject**
  Description: Returns the low level instance.
  Syntax: CMPIEnumeration* getLowLevelObject (void) const;
  Returns: Low level instance

- **getNext**
  Description: Get the next element of this enumeration.
  Syntax: CCIMData getNext (void) const;
  Returns: The next element

- **hasNext**
  Description: Test for any elements left in this enumeration.
  Syntax: bool hasNext (void) const;
  Returns: true if any elements left else false

4.7 CCIMArgument

A class representing a CIM method argument.

Member Functions:

- clone
- addArg

- getArg
- getArgCount
- getArgAt
- getLowLevelObject

Static Member Functions:

- toCCIMArgument
- create
- toCMPIValue

Constructor Description:

Description: Constructor taking CCIMArgument as argument
Syntax: CCIMArgument (const CCIMArgument& ca);
Parameters:
- *ca* CCIMArgument

Member Function Description:

- **toCCIMArgument**
  Description: Convert the low level Argument to CCIMArgument. The low level argument should not be released or assigned to another object after this call. We could have cloned this low level argument but this would incur an unnecessary memory allocation since most of the use cases will be just a straight conversion from low level argument.
  Syntax: static CCIMArgument toCCIMArgument (CMPIArgs* args, bool auto_release = true);
  Parameters:
  - *args* low level Argument

- **create**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMArgument create (CMPIValue* val, bool auto_release);
  Parameters:
  - *val* low level CMPIValue
  Returns: CCIMArgument

- **toCMPIValue**
  Description: Converts CCIMArgument value to a low level CMPIValue object.
  Syntax: static CMPIValue toCMPIValue (CCIMArgument val)
  Parameters:
  - *val* CCIMArgument value

Returns: low level CMPIValue object

- **clone**
  Description: Clone the object.
  Syntax: CCIMArgument* clone (void) const;
  Returns none

- **addArg**
  Description: Adds / replaces a named argument.
  Syntax: void addArg (const string& name, CCIMValue value);
  Parameters:
  - *name* Name of the argument to add
  - *value* CCIMValue to add or replace the named argument

- **getArg**
  Description: Gets a named argument value.
  Syntax: CCIMData getArg (const string& name) const;
  Parameters:
  - *name* name of the argument to get
  Returns The argument

- **getArgCount**
  Description: Gets the number of arguments contained in this Args.
  Syntax: unsigned int getArgCount (void) const;
  Returns The number of argument

- **getArgAt**
  Description: Gets a argument value defined by its index.
  Syntax: CCIMData getArgAt (unsigned int index, string* name) const;
  Parameters:
  - index index of the argument to get
  - name name of the argument to get
  Returns: The argument

- **getLowLevelObject**
  Description: Returns the low level argument.
  Syntax: CMPIArgs* getLowLevelObject (void) const;
  Returns The low level argument

4.8 CCIMArray

An Array holding CIM types.

Member Functions:

- getSize
- getElementAt
- setElementAt
- clone
- getLowLevelObject

Static Member Functions:

- toCCIMArray
- create
- toCMPIValue

Constructor Description:

- **CCIMArray**
  Description: Constructor taking CCIMArray as argument
  Syntax: CCIMArray (const CCIMArray& ca);
  Parameters:
  - *ca* CCIMArray

Member Function Descriptions:

- **toCCIMArray**
  Description: Make a CCIMArray from the underlying low level object.
  Syntax: static CCIMArray toCCIMArray (CMPIArray* array, bool auto_release = true);
  Parameters:
  - *array* Low Level object

- **create**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMArray create (CMPIValue* val, bool auto_release);
  Parameters:
  - *val* Low level CMPIValue

- **toCMPIValue**
  Description: Converts CCIMArray val to a low level CMPIValue object.
  Syntax: static CMPIValue toCMPIValue (CCIMArray val);
  Parameters:
  - *val* CCIMArray value

- **getSize**
  Description: Gets the number of elements contained in this array.
  Syntax: unsigned int getSize (void) const;
  Returns: The size of the array

- **getElementAt**
  Description: Gets an element value defined by its index.
  Syntax: CCIMData getElementAt (unsigned int index) const;
  Parameters:
    - *index* index of the element to get

  Returns The Element

- **setElementAt**
  Description: Sets an element value defined by its index.
  Syntax: void setElementAt (unsigned int index, CCIMValue value);
  Parameters:
    - *index* index of the element to set
    - *value* CCIMValue to set

- **clone**
  Description: Clone this array.
  Syntax: CCIMArray* clone (void) const;
  Returns none

- **getLowLevelObject**
  Description: Returns the low level object
  Syntax: CMPIArray* getLowLevelObject (void) const;
  Returns: Array of objects

4.9 CCIMString

A Class representing a CIM String.

Member Functions:

- clone
- getLowLevelObject

Static Member Functions:

- toCCIMString
- create
- toCMPIValue

Constructor Description:

- **CCIMString**
  Description: Constructor to construct from a string.
  Syntax: CCIMString (string str);
  Parameters:
  - *str* String

Member Function Description:

- **toCCIMString**
  Description: Convert the low level string to CCIMString. The low level string should not be released or assigned to another object after this call. We could have cloned this low level string but this would incur an unnecessary memory allocation since most of the use cases will be just a straight conversion from low level argument.
  Syntax: static CCIMString toCCIMString (CMPIString* str, bool auto_release = true);
  Parameters:
  - *str* low level string

- **create**
  Description: Creates this object from a low level CMPIValue.
  Syntax: static CCIMString create (CMPIValue* val, bool auto_release)
  Parameters:
  - *val* low level CMPIValue.

- **toCMPIValue**
  Description: Converts CCIMString value to a low level CMPIValue object.
  Syntax: static CMPIValue toCMPIValue (CCIMString val);
  Parameters:
  - *val* CCIMString value
  Returns: low level CMPIValue object.

- **clone**
  Description: Clone this object.
  Syntax: CCIMString* clone (void) const;
  returns: none

- **getLowLevelObject**

Description:	Returns the low level object
Syntax:	CMPIString* getLowLevelObject (void) const;
Returns:	the low level object

5 High Level API-C

This is an abstraction of the C++ high level Application Programming Interface in C language. The objects in this interface consist of an opaque pointer (this corresponds to the C++ objects) and a corresponding function table (this corresponds to the objects access functions). The functions in the function table delegate the calls to the appropriate C++ functions. It has the same set of API's in C++ API. The user can do the same operations of C++ API in C.

5.1. Connection API's

- 5.1.1 Discoverer
- 5.1.2 CIMMAP
- 5.1.3 Subject

5.2. Component API's

- 5.2.1 ComputerSystem
- 5.2.2 Processor
- 5.2.3 ProcessorCore
- 5.2.4 PhysicalMemory
- 5.2.5 PhysicalAsset
- 5.2.6 Fan
- 5.2.7 Sensor
- 5.2.8 Software
- 5.2.9 BootConfig
- 5.2.10 User
- 5.2.11 PowerSupply
- 5.2.12 FanRedundancySet
- 5.2.13 PowerSupplyRedundancySet
- 5.2.14 Battery
- 5.2.15 BIOSManagement
- 5.2.16 DHCPClient
- 5.2.17 DNSClient
- 5.2.18 IPInterface
- 5.2.19 NetworkPort
- 5.2.20 OpaqueManagementData
- 5.2.21 OperatingSystem
- 5.2.22 TextRedirection
- 5.2.23 USBRedirection

5.2.24 VirtualMedia
5.2.25 EthernetPort
5.2.26 RegisteredProfile
5.2.27 Error Functions

5.1. Connection API's

5.1.1. Discoverer

- **discoverMAPs**

  Description: Discover the management access point in the network range between start_ip and end_ip

  Syntax:
```
void discoverMAPs (char* start_ip,
                   char* end_ip,
                   PortScheme_T* port_schemes,
                   int num_port_schemes,
                    u32 timeout,
                   int max_maps,
                   DSDKCIMMAP* cimmap[],
                   int* num_maps);
```

  Parameters:
  - *start_ip* Starting IP Address from which to search from
  - *end_ip* Ending IP Address at which the search should end. If this value is empty then only the start_ip is searched.
  - *port_schemes* Array of structure PortScheme_T (Port numbers and HTTP protocol /scheme). If this is NULL default ports 623(http) and 664 (https) will be used.
  - *num_port_schemes* Number of ports/schemes in 'port_schemes' array.
  - *timeout* Timeout.
  - *max_maps* Size of cimmap array
  - *cimmap* Contain the list of the discovered CIM MAPs after execution of this function.
  - *num_maps* Total number of MAP's discovered.

- **discoverMAP**

  Description: Discover the management access point that is present in host_name.

  Syntax:
```
void discoverMAP (char* host_name,
                  char* port,
                  char* http_scheme,
                  u32 timeout,
                  int max_maps,
                  DSDKCIMMAP* cimmap [],
                  int* num_maps);
```

  Parameters:

- *host_name* Host name to get the MAP.
- *port* *Port number,* when not specified uses the default port (623 &664).
- *http_scheme* Http Protocol.If it is not specified http protocol will be used by default.If it is blank (""),default protocol for the port will be used.(https for 664 and http for other ports including 623).
- *timeout* Timeout
- *max_maps* Size of cimmap array
- *cimmap* Contain the list of the discovered CIM MAPs after execution of this function.
- *num_maps* Total number of MAP's discovered.

This could returns more than one MAP on same IP address as there could more than one MAP at different ports.

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- discoverer_c.h
Library -- dashapic

Usage Examples

Below is an example on how this function can be used by an application. This example discovers the the management access point from the IP address 192.168.1.2 to 192.168.1.5 on ports 623, 664 .

```
#include <stdio.h>
#include "subject_c.h"
#include "cimmap_c.h"
#include "discoverer_c.h"


int
main (int arg, char * argv [])
{
        DSDKCIMMAP* cimmap;
        int nummap;
        port_scheme_T portschemeshttp, portschemeshttps;
```

```
        portschemeshttp.port            = "623";
        portschemeshttp.http_scheme     = "http";

        discoverMAPs  ("192.168.1.2","192.168.1.5",&portschemeshttp,1,5,&cimmap,&nummap);

        portschemeshttps.port           = "664";
        portschemeshttps.http_scheme    = "https";

        discoverMAPs  ("192.168.1.2","192.168.1.5",&portschemeshttps,1,5,&cimmap,&nummap);


        return 0;
}
```

Below is a sample out put for above example

```
Discovered :
                192.168.1.3:623
                192.168.1.3:664
```

Usage Examples

Below is an example on how this function can be used by an application. This example discovers the the management access point from the IP address 192.168.0.11 to 192.168.0.11 on ports 623, 664, 16992

```
DSDKCIMMAP** maps;/* to be filled in with pointer to discovered maps */
int mapcount;           /* to be filled in with # of discovered maps */
int errorcode, i;
PortScheme_T ports[3];
char hostname[100]; char port[100];

ports[0].port = "623";          ports[0].http_scheme = "http";
ports[1].port = "664";          ports[1].http_scheme = "https";
ports[2].port = "16992";  ports[2].http_scheme = "http";

maps = malloc ((sizeof (DSDKCIMMAP*)) * MAX_MAP_COUNT);

/* do the discovery */

discoverMAPs ("192.168.0.11", /* starting address */
"192.168.0.11",                 /* ending address */
ports,                          /* list of ports to check */
3,                                      /* number of ports in the list */
2,                                      /* timeout in seconds */
MAX_MAP_COUNT,                  /* maximum maps can be discovered */
```

```
maps,                           /* where to put the discovered maps */
&mapcount                               /* where to put the # of discovered maps */
);
```

Below is a sample out put for above example

```
Discovered :
                192.168.0.11:623
                192.168.0.11:664
```

5.1.2. CIMMAP

- **DSDKCIMMAP.**
  A structure representing CIMMAP.

  Structure Members
  - hdl - Opaque pointer to CIMMAP specific implementations
  - ft - Pointer to CIMAP function table.

- **makeCIMMAP**
  Description: The function creats and return a DSDKCIMMAP structure pointer.
  Syntax: DSDKCIMMAP* makeCIMMAP (const char* host_name,
  const char* port);
  Parameters:
  - *host_name* Host name
  - *port* Port Number, when not specified uses the default port(8888).

  Returns: The pointer to DSDKCIMMAP, the pointer should be released by release function of DSDKCIMMAPFT.

- **DSDKCIMMAPFT**
  A structure representing CIMMAP function table.

Member function

- connect
- getHostName
- getPort
- getLastError
- release

Member Functions Description

- **connect**

  Description: Connects to a DASH Server using the credentials supplied in subject.

  Syntax:
  ```
  DSDKClient* (*connect)(DSDKCIMMAP* map,
                         DSDKSubject* sub)
  ```

  Parameters:
  - *map* Pointer to CIMMAP
  - *sub* Credentials to connect.
  - *max_len* Maximum length of the buffer

  Returns: A client handle.

- **getHostName**

  Description: Gets the host name

  Syntax:
  ```
  void (*getHostName)(DSDKCIMMAP *map,  char *host_name,
                      int max_len);
  ```

  Parameters:
  - *map* Pointer to CIMMAP
  - *host_name* Pointer to buffer that receives the host name
  - *max_len* Maximum length of the buffer

- **getPort**

  Description: Gets the port

  Syntax:
  ```
  void (*getPort)(DSDKCIMMAP *map,
                  char *port,
                  int max_len) ;
  ```

  Parameters:
  - *map* Pointer to CIMMAP
  - *port* Pointer to buffer that receives the port
  - *max_len* Maximum length of the buffer

- **getLastError**

  Description: Gets the last error

  Syntax:
  ```
  unsigned int (*getLastErro)(DSDKCIMMAP *map);
  ```

  Parameters:
  - *map* Pointer to CIMMAP

  Returns: Last error

- **release**

  Description: Release this object.

  Syntax:
  ```
  unsigned int (*release)(DSDKCIMMAP *map);
  ```

  Parameters:
  - *map* Pointer to CIMMAP

Note: All functions in C library should call ***dsdkc_getLastError*** function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- cimmap_c.h
Library -- dashapic

Usage Examples

```
main ()
{
        DSDKSubject*                    subject;
        DSDKCIMMAP*                     cimmap;
        DSDKClient*                     client;

        subject = makeSubject ("admin", "admin", "basic", 0, 0, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, 0);
        cimmap  = makeCIMMAP ("192.168.0.20", "623");
        client  = cimmap->ft->connect (cimmap, subject);


        /* ------ do your stuff --------*/


        subject->ft->release (subject);
        cimmap->ft->release (cimmap);
}
```

5.1.3. Subject

- **DSDKSubject**
  A structure representing a subject.

  Structure Members
  - hdl - Opaque pointer to Subject specific implementations
  - ft - Pointer to Subject function table.

- **makeSubject**

Description: The function creats and return a DSDKSubject structure pointer.

Syntax:
```
DSDKSubject* makeSubject (const char*   user,
                          const char*   password,
                          const char*   auth,
                                int     verify_peer,
                                int     verify_host,
                          const char*   ca_info,
                          const char*   ca_path,
                          const char*   cert_file,
                          const char*   key_file,
                          const char*   proxy,
                          const char*   proxy_user,
                          const char*   proxy_password,
                          unsigned long timeout);
```

Returns: The pointer to DSDKSubject, the pointer should be released by release function of DSDKSubjectFT.

- **DSDKSubjectFT**
  A structure representing the Subject function pointer table.

Member Function

Member Functions Description

- **getLastError**
  Description: Gets the last error
  Syntax: unsigned int (*getLastErro)(DSDKSubject *sub);
  Parameters:
  - *sub* Pointer to DSDKSubject

- **release**
  Description: Release this object.
  Syntax: unsigned int (*release)(DSDKSubject *sub);
  Parameters:
  - *sub* Pointer to DSDKSubject

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- subject_c.h
Library -- dashapic

5.2. Component Classes

5.2.1. ComputerSystem

- **DSDKComputerSystemIterator**

  A structure representing a computer system iterator.

  Structure Members
  - hdl - Opaque pointer to computer system specific implementations
  - ft - Pointer to computer system iterator function table.

- **enumComputerSystems**

  Description: Enumerates all the computer system present under a management access point.

  Syntax: DSDKComputerSystemIterator* enumComputerSystems (DSDKClient* client, BOOL cached);

  Parameters:
  - *client* Pointer to the client interface.
  - *Cached*Enable/Disable caching.

  Returns: The computer system iterator.

- **ComputerSytemIteratorFt**

  A structure representing computer system iterator function table.

Member Functions

- getItem
- isEnd
- next
- release
- getTestName

Member Functions Description

- **getItem**

  Description: Gets the computer system at this iterator location.

  Syntax: DSDKComputerSystem*(*getItem)(DSDKComputerSystemIterator *di);

  Parameters:
  - *di* Computer system iterator.

Returns: The computer system at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKComputerSystemIterator *di);

  Parameters:

- *di* Computer system iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKComputerSystemIterator *di);

  Parameters:

- *di* Computer system iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKComputerSystemIterator *di);

  Parameters:

  - *di* Computer system iterator.

- **getTestName**

  Description: getTestName

  Syntax: void (*getTestName) (DSDKComputerSystemIterator* cs, char* name, int max_len);

  Parameters:

- *cs* Computer system iterator function table .
  - *name* Pointer to buffer that receives the name
  - *max_len* Maximum length of the buffer

- **DSDKComputerSystem**

  A structure representing a computer system

  Structure Members

  - hdl - Opaque pointer to computer system specific implementations
  - ft - Pointer to computer system function table.

- **DSDKComputerSytemFT**

  A structure representing computer system function table.

Member Functions

- getName
- getPrimaryOwner
- getPrimaryOwnerContact
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getDedicated
- getDedicatedStr
- getPowerState
- getPowerStateStr
- getRequestedPowerState
- getRequestedPowerStateStr
- getPowerChangeCapabilities
- getPowerChangeCapabilitiesStr
- getPowerStatesSupported
- getPowerStatesSupportedStr
- powerOn
- powerOff
- powerCycle
- powerReset
- getReqPwrStateChangeErrStr
- release

Member Functions Description

- **getName**

  Description: Returns the name of the computer system

  Syntax: void(* getName)(DSDKComputerSystem *cs, char *name, int max_len);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *name* Pointer to buffer that receives the name
  - *max_len* Maximum length of the buffer

- **getPrimaryOwner**

  Description: Gets the primary owner of the computer system

  Syntax: void (*getPrimaryOwner) (DSDKComputerSystem* cs, char* owner, int max_len);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *name* Pointer to buffer that receives the name
  - *max_len* Maximum length of the buffer

- **getPrimaryOwnerContact**

  Description: Gets the primary owner contact of the computer system

  Syntax: void(* getPrimaryOwnerContact)(DSDKComputerSystem *cs, char *contact, int max_len);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *owner* Pointer to buffer that receives the owner contact
  - *max_len* Maximum length of the buffer

- **getEnabledState**

  Description: Gets the EnabledState of the computer system

  Syntax: uint16 (*getEnabledState) (DSDKComputerSystem* cs);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem

  Returns: The EnabledState

- **getEnabledStateStr**

  Description: Gets the enabled state of the computer system as string

  Syntax: void (*getEnabledStateStr) (DSDKComputerSystem* cs, char* str, int max_len);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *str* Pointer to buffer that receives the name
  - *max_len* Maximum length of the buffer

  Returns: The EnabledState

- **getRequestedState**

  Description: Gets the last RequestedState of the computer system

  Syntax: uint16 (*getRequestedState) (DSDKComputerSystem* cs);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem

  Returns: The RequestedState

- **getRequestedStateStr**

  Description: Gets the last Requested state of the computer system as string

  Syntax: void (*getRequestedStateStr) (DSDKComputerSystem* cs, char* str, int max_len);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - str Pointer to buffer that receives the name

- max_len Maximum length of the buffer

- **getDedicated**
  Description: Gets the purpose(s) of this computer is dedicated to.
  Syntax: int (*getDedicated) (DSDKComputerSystem* cs, uint16* dedicated, int max_dedicated);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - dedicated The dedicated purpose(s) is filled in here.
  - max_dedicated Maximum dedicated purpose to be filled in dedicated.

- **getDedicatedStr**
  Description: Gets the purpose(s) of this computer is dedicated to as string
  Syntax: int (*getDedicatedStr) (DSDKComputerSystem* cs, char** dedicated, int max_dedicated, int max_strlen);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - dedicated The dedicated purpose(s) is filled in here.
  - max_dedicated Maximum dedicated purpose to be filled in dedicated.
  - max_strlen Maximum buffer size for dedicated.

- **getPowerState**
  Description: Gets the current power state of this computer system.
  Syntax: uint16(* getPowerState)(DSDKComputerSystem *cs);
  Returns: The power state.

- **getPowerStateStr**
  Description: Gets the power state as string.
  Syntax: void(*getPowerStateStr)(DSDKComputerSystem *cs, char *str, int max_len);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *str* Pointer to buffer that receives the power state
  - *max_len* Maximum length of the buffer

  Returns: The EnabledState

- **getRequestedPowerState**
  Description: Gets the Requested Power State of this computer system.
  Syntax: uint16 (*getRequestedPowerState) (DSDKComputerSystem* cs);

Parameters:

- *cs* Pointer to DSDKComputerSystem

- **getRequestedPowerStateStr**

  Description: Gets the Requested power state of this computer system as a string.

  Syntax: void (*getRequestedPowerStateStr) (DSDKComputerSystem* cs, char*str, int max_len);

  Parameters:

  - *cs* Pointer to DSDKComputerSystem
  - *str* Pointer to buffer that receives the power state
  - *max_len* Maximum length of the buffer

  Returns: The EnabledState

- **getPowerChangeCapabilities**

  Description: Gets the power change capabilities of this computer system.

  Syntax: int (*getPowerChangeCapabilities) (DSDKComputerSystem* cs, uint16* power_change, int max_types);

  Parameters:

  - *cs* Pointer to DSDKComputerSystem
  - *power_change* The power change capabilities
  - *max_types* maximum types

- **getPowerChangeCapabilitiesStr**

  Description: Gets the power change capabilities of this computer system as string

  Syntax: int (*getPowerChangeCapabilitiesStr) (DSDKComputerSystem* cs, char** capabilities, int max_capabilities, int max_strlen);

  Parameters:

  - *cs* Pointer to DSDKComputerSystem
  - *capabilities* The power capabilities(s) is filled in here.
  - *max_capabilities* Maximum capabilities purpose to be filled in capabilities.
  - *max_strlen* Maximum buffer size for capabilities.

- **getPowerStatesSupported**

  Description: Gets the power states supported for this computer system.

  Syntax: int (*getPowerStatesSupported) (DSDKComputerSystem* cs, uint16* power_states, int max_types);

  Parameters:

  - *cs* Pointer to DSDKComputerSystem
  - *power_states* The power states supported

- *max_types* maximum types

- **getPowerStatesSupportedStr**
  Description: Gets the power states supported for this computer system.
  Syntax: int (*getPowerStatesSupportedStr) (DSDKComputerSystem* cs, char**power_states, int max_states, int max_strlen);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *power_states* The power states(s) is filled in here.
  - *max_states* Maximum power state to be filled in power_states.
  - *max_strlen* Maximum buffer size for power_states.

- **powerOn**
  Description: Power on the computer system
  Syntax: uint32(*powerOn) (DSDKComputerSystem *cs);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem

  Returns: 0 if success, else return error code.

- **poweroff**
  Description: Power off the computer system
  Syntax: uint32(*powerOff) (DSDKComputerSystem *cs);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - Returns: 0 if success, else return error code.

- **powerCycle**
  Description: Power cycle the computer system
  Syntax: uint32(* powerCycle) (DSDKComputerSystem *cs);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem

  Returns: 0 if success, else return error code.

- **powerReset**
  Description: Power reset the computer system
  Syntax: uint32(*powerReset) (DSDKComputerSystem *cs);
  Parameters:
  - *cs* Pointer to DSDKComputerSystem

  Returns: 0 if success, else return error code.

- **getReqPwrStateChangeErrStr**

Description: Gets the error description for request change power state error code.

Syntax: void getReqPwrStateChangeErrStr (DSDKComputerSystem *cs, uint32 err, char*err_str, int max_len);

Parameters:

- *cs* Pointer to DSDKComputerSystem
- *err* Error code
- *err_str* Pointer to buffer to receive error description.
- *max_len* Maximum length of the buffer.

- **release**

Description: Releases this object

Syntax: void(* release)(DSDKComputerSystem *cs);

Parameters:

- *cs* Pointer to DSDKComputerSystem

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- computersystem_c.h
Library -- dashapic

Usage Examples

Example:

```
main ()
{
        DSDKSubject*                    subject;
        DSDKCIMMAP*                             cimmap;
        DSDKClient*                     client;
        DSDKComputerSystemIterator* iter;



        subject = makeSubject ("admin", "admin", "basic", 0, 0, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, 0);
        cimmap  = makeCIMMAP ("192.168.0.20", "623");
        client    = cimmap->ft->connect (cimmap, subject);

        /* enumerate the computer system */
        iter  = enumComputerSystems (client, 1);
```

```
while (!iter->ft->isEnd (iter))
{
        /* get the instance */
        DSDKComputerSystem* cs = iter->ft->getItem (iter);
        if (cs != 0)
        {
                char name [257];
                cs->ft->getName (cs, name, 256);

                printf ("%s\n", name);
        }

        /* release the instance */
        cs->ft->release (cs);

        /* iterate to next instance */
        iter->ft->next (iter);
}

iter->ft->release (iter);
subject->ft->release (subject);
cimmap->ft->release (cimmap);
```

5.2.2. Processor

- **DSDKProcessorIterator**

  A structure representing a processor iterator..

  Structure Members
  - hdl - Opaque pointer to processor specific implementations
  - ft - Pointer to processor iterator function table.

- **enumProcessors**

  Description: Enumerate all the processors present under a management access point.

  Syntax: DSDKProcessorIterator* enumProcessors (DSDKClient* client, BOOL cached);

  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Processor iterator.

- **ProcessorIteratorFt**

A structure representing processor iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the processor at this iterator location.

  Syntax: DSDKProcessor*(*getItem)(DSDKProcessorIterator *di);

  Parameters:

- *di* Processor iterator.

  Returns: The processor at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKProcessorIterator *di);

  Parameters:

- *di* Processor iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKProcessorIterator *di);

  Parameters:

- *di* Processor iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKProcessorIterator *di);

  Parameters:

- *di* Processor iterator.

- **DSDKProcessor**

  A structure representing a Processor.

  Structure Members

  - hdl - Opaque pointer to Processor specific implementations
  - ft - Pointer to Processor function table.

- **DSDKProcessorFT**
  A structure representing processor function table.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getFamily
- getCurrentClockSpeed
- getMaxClockSpeed
- getExternalBusClockSpeed
- getCPUStatus
- getCPUStatusStr
- getLoadPercentage
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getElementName
- getOtherFamilyDescription
- release

Member Functions Description

- **getSystemCreationClassName**
  Description: Gets the processor System Creation class Name
  Syntax: void (*getSystemCreationClassName) (DSDKProcessor* proc, char* name, int max_len);
  Parameters:
  - *proc* Pointer to DSDKProcessor
  - *name* Creation Class Name of the processor
  - *max_len* maximum buffer length

  Returns: none

- **getSystemName**
  Description: Gets SystemName of the processor

Syntax: void (*getSystemName) (DSDKProcessor* proc, char* name, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor
- *name* system Name of the processor
- *max_len* maximum buffer length

returns: none

- **getCreationClassName**

Description: Gets the processor Creation class Name

Syntax: void (*getCreationClassName) (DSDKProcessor* proc, char* name, int max_len);

Parameters:

- proc Pointer to DSDKProcessor
- name Creation Class Name of the processor
- max_len maximum buffer length

Returns: none

- **getDeviceID**

Description: Gets the processor device id.

Syntax: void (*getDeviceID) (DSDKProcessor* proc, char* dev_id, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor
- *dev_id* Buffer to store device ID
- *max_len* maximum buffer length

Returns: none

- **getFamily**

Description: Gets the processor family

syntax: void (*getFamily) (DSDKProcessor* proc, char* method, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor
- *method* Buffer to store Family
- *max_len* maximum buffer length

Returns: none

- **getCurrentClockSpeed**

Description: Gets the current clock speed

Syntax: uint32 (*getCurrentClockSpeed) (DSDKProcessor* proc);

Parameters:

- *proc* Pointer to DSDKProcessor

Returns: The current clock speed in Mhz

- **getMaxClockSpeed**

  Description: Gets the maximum clock speed

  Syntax: uint32 (*getMaxClockSpeed) (DSDKProcessor* proc);

  Parameters:

  - *proc* Pointer to DSDKProcessor

  Returns: The maximum clock speed in Mhz.

- **getExternalBusClockSpeed**

  Description: Gets the external bus clock speed

  Syntax: uint32 (*getExternalBusClockSpeed) (DSDKProcessor* proc);

  Parameters:

  - *proc* Pointer to DSDKProcessor

  Returns: The external bus clock speed.

- **getCPUStatus**

  Description: Gets the CPU Status

  Syntax: uint16 (*getCPUStatus) (DSDKProcessor* proc);

  Parameters:

  - *proc* Pointer to DSDKProcessor

  Returns: The CPU Status .

- **getCPUStatusStr**

  Description: Gets the CPU status of the processor as string

  Syntax: void (*getCPUStatusStr) (DSDKProcessor* proc, char* status, int max_len);

  Parameters:

  - *proc* Pointer to DSDKProcessor
  - *status* The Status of cpu is filled in
  - *max_len* maximum buffer length

  Returns: none

- **getLoadPercentage**

  Description: Gets the load percentage of this processor

  Syntax: uint16 (*getLoadPercentage) (DSDKProcessor* proc);

  Parameters:

  - *proc* Pointer to DSDKProcessor

  Returns: The load percentage of this processor.

- **getEnabledState**

  Description: Gets the state of the processor

Syntax: uint16 (*getEnabledState) (DSDKProcessor* proc);

Parameters:

- *proc* Pointer to DSDKProcessor

Returns: The enabled state

- **getEnabledStateStr**

  Description: Gets the state of the processor as string

  Syntax: void (*getEnabledStateStr) (DSDKProcessor* proc, char* state, int max_len);

  Parameters:

  - *proc* Pointer to DSDKProcessor
  - *state* The Enabled State is filled in
  - *max_len* maximum buffer length

  Returns: none

- **getRequestedState**

  Description: Gets the requested state of the processor

  Syntax: uint16 (*getRequestedState) (DSDKProcessor* proc);

  Parameters:

  - *proc* Pointer to DSDKProcessor

  Returns: The requested state

- **getRequestedStateStr**

  Description: Gets the requested status of the processor as string

  Syntax: void (*getRequestedStateStr) (DSDKProcessor* proc, char* state, int max_len);

  Parameters:

  - *proc* Pointer to DSDKProcessor
  - *state* The requested state is filled in.
  - *max_len* maximum buffer length

  Returns: none

- **getOperationalStatus**

  Description: Gets the list of OperationalStatus

  Syntax: int (*getOperationalStatus) (DSDKProcessor* proc, uint16* op_status, int max_types);

  Parameters:

  - *proc* Pointer to DSDKProcessor
  - *op_status* Operational Status
  - *max_types* maximum types

  Returns: List of operational status

- **getOperationalStatusStr**

Description: Gets the list of operational status as string

syntax: int (*getOperationalStatusStr) (DSDKProcessor* proc, char** os, int max_os, int max_strlen);

Parameters:

- *proc* Pointer to DSDKProcessor
- *os* The operational status is filled in.
- *max_os* maximum number of operational status
- *max_strlen* maximum buffer size

Returns: none

- **getHealthState**

Description: Gets the health state of the processor

Syntax: uint16 (*getHealthState) (DSDKProcessor* proc);

Parameters:

- *proc* Pointer to DSDKProcessor

Returns: The health state

- **getHealthStateStr**

Description: Gets the health state of the processor as string

Syntax: void (*getHealthStateStr) (DSDKProcessor* proc, char* state, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor
- *state* The health state is filled in.
- *max_len* maximum buffer length

Returns: none

- **getElementName**

Description: Gets the Element Name of the processor

Syntax: void (*getElementName) (DSDKProcessor* proc, char* name, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor
- *name* The Element Name
- *max_len* maximum buffer length

Returns: none

- **getOtherFamilyDescription**

Description: Gets the Other FamilyD escription of the processor

Syntax: void (*getOtherFamilyDescription) (DSDKProcessor* proc, char* desc, int max_len);

Parameters:

- *proc* Pointer to DSDKProcessor

- *name* The OtherFamilyDescription
- *max_len* maximum buffer length

Returns none

- **release**

  description: Releases this object

  Syntax: void (*release) (DSDKProcessor* proc);

  Returns: none

Note: All functions in C library should call dsdkc_getLastError function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use dsdkc_getLastErrorStr to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- processor_c.h
Library -- dashapic

5.2.3. ProcessorCore

- **DSDKProcessorCoreIterator**

  A structure representing a processor core iterator.

  Structure Members

  - hdl - Opaque pointer to processor core specific implementations
  - ft - Pointer to processor core iterator function table.

- **enumProcessorCores**

  Description: Enumerate all the processor cores present under a management access point.

  Syntax: DSDKProcessorCoreIterator* enumProcessorCores (DSDKClient* client,BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Processor core iterator.

- **ProcessorCoreIteratorFt**

  A structure representing processor core iterator function table.

Member Functions

- getItem

- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the processor core at this iterator location.

  Syntax: DSDKProcessorCore*(*getItem)(DSDKProcessorCoreIterator *di);

  Parameters:

- *di* Processor core iterator.

  Returns: The processor core at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKProcessorCoreIterator *di);

  Parameters:

- *di* Processor core iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKProcessorCoreIterator *di);

  Parameters:

- *di* Processor core iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKProcessorCoreIterator *di);

  Parameters:

- *di* Processor core iterator.

- **DSDKProcessorCoreFT**

  A structure representing processor core function table.

Member Functions

- getItem
- getInstanceID
- getCoreEnabledState
- getEnabledState
- getLoadPercentage
- getRequestedState

- getOperationalStatus
- getHealthState
- getElementName
- release

Member Functions Description

- **getInstanceID**

  Description: Gets the instance ID of the processor Core

  Syntax: void (*getInstanceID) (DSDKProcessorCore* pc, char* instid, int max_len);

  Parameters:

  - *pc* pointer to processor core
  - *instid* The Instance ID
  - *max_len* maximum buffer length

- **getCoreEnabledState**

  Description: Gets the Core Enabled state of processor Core

  Syntax: uint16 (*getCoreEnabledState) (DSDKProcessorCore* pc);

  Parameters:

  - *pc* pointer to processor core

  Returns: The Core Enabled state

- **getEnabledState**

  Description: Gets the Enabled state of processor Core

  Syntax: uint16 (*getEnabledState) (DSDKProcessorCore* pc);

  Parameters:

  - *pc* pointer to processor core

  Returns: The Enabled state

- **getLoadPercentage**

  Description: Gets the load percentage of this processor core

  Syntax: uint16 (*getLoadPercentage) (DSDKProcessorCore* pc);

  Parameters:

  - *pc* pointer to processor core

  Returns: The load percentage of this processor core.

- **getRequestedState**

  Description: Gets the Requested State state of processor Core

  Syntax: uint16 (*getRequestedState) (DSDKProcessorCore* pc);

  Parameters:

  - *pc* pointer to processor core

  Returns: The Requested State

- **getOperationalStatus**

  Description: Gets the OperationalStatus

  Syntax: int (*getOperationalStatus) (DSDKProcessorCore* pc, uint16* op_status, int max_types);

  Parameters:

  - *pc* pointer to processor core
  - *op_status* Operational Status
  - *max_types* maximum types

- **getHealthState**

  Description: Gets the health state of the processor

  Syntax: uint16 (*getHealthState) (DSDKProcessorCore* pc);

  Parameters:

  - *pc* pointer to processor core

  Returns: Health state

- **getElementName**

  Description: Gets the Element Name of the processor Core

  Syntax: void (*getElementName) (DSDKProcessorCore* pc, char* name, int max_len);

  Parameters:

  - *pc* pointer to processor core
  - *name* The Element Name
  - *max_len* maximum buffer length

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKProcessorCoreIterator *di);

  Parameters:

  - *pc* pointer to processor core

- **DSDKProcessorCoreFT**

  A structure representing processor core function table.

Member Functions

- getCharacteristics
- getInstanceID
- getLoadPercentage
- getStatus
- release

Member Functions Description

- **getCharacteristics**

  Description**:** Gets the processor core characteristics

  Syntax: void(* getCharacteristics)(DSDKProcessorCore *pc, uint16 **characteristics, int max_characteristics);

- **getInstanceID**

  Description: Gets the instance ID of the processor Core

  Syntax: void(*getInstanceID)(DSDKProcessorCore *pc, char *instid, int max_len);

  Parameters:

  - *pc* Pointer to DSDKProcessorCore
  - *stepping* Pointer to buffer that receives the instid
  - *max_len* Maximum length of buffer

- **getLoadPercentage**

  Description**:** Gets the load percentage of this processor core

  Syntax: uint16(*getLoadPercentage)(DSDKProcessorCore *pc);

  Parameters:

  - *pc* Pointer to DSDKProcessorCore

  Returns: The load percentage of this processor core.

- **getStatus**

  Description**:** Gets the current status of the core

  Syntax: uint16(* getStatus)(DSDKProcessorCore *pc);

  Parameters:

  - *pc* Pointer to DSDKProcessorCore

  Returns: The current status of the core.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKProcessorCore *pc);

  Parameters:

  - *pc* Pointer to DSDKProcessorCore

Note: All functions in C library should call ***dsdkc_getLastError*** function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use ***dsdkc_getLastErrorStr*** to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- processor_c.h

Library -- dashapic

5.2.4. Memory

- **DSDKMemoryIterator**

  A structure representing a memory iterator.

  Structure Members

  - hdl - Opaque pointer to memory specific implementations
  - ft - Pointer to memory iterator function table.

- **enumComputerSystems**

  Description: Enumerate all the physical memory present under a management access point.

  Syntax: DSDKMemoryIterator* enumlMemory (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The physical memory iterator.

- **MemoryIteratorFt**

  A structure representing memory iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the memory at this iterator location.

  Syntax: DSDKMemory*(*getItem)(DSDKlMemoryIterator *di);

  Parameters:

  - *di* memory iterator.

  Returns: The memory at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKMemoryIterator *di);

Parameters:

- *di* memory iterator.

Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKMemoryIterator *di);

  Parameters:

- *di* memory iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKMemoryIterator *di);

  Parameters:

- *di* memory iterator.

- **DSDKComputerSystem**

  A structure representing a memory

  Structure Members

  - hdl - Opaque pointer to memory specific implementations
  - ft - Pointer to memory function table.

- **DSDKMemoryFT**

  A structure representing memory function table.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- isVolatile
- getAccess
- getAccessStr
- getBlockSize
- getNumberOfBlocks
- getConsumableBlocks
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr

- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getElementName
- release

Member Functions Description

- **getSystemCreationClassName**

  Description: Gets the memory System Creation class Name

  Syntax: void (*getSystemCreationClassName) (DSDKMemory* pm, char* name,int max_len);

  Parameters:
  - *pm* pointer to memory
  - *name* Creation Class Name of the memory
  - *max_len* maximum buffer length

- **getSystemName**

  Description: Gets SystemName of the memory

  Syntax: void (*getSystemName) (DSDKMemory* pm, char* name, int max_len);

  Parameters:
  - *pm* pointer to memory
  - *name* system Name of the memory
  - *max_len* maximum buffer length

- **getCreationClassName**

  Description: Gets the memory Creation class Name

  Syntax: void (*getCreationClassName) (DSDKMemory* pm, char* name, int max_len);

  Parameters:
  - *pm* pointer to memory
  - *name* Creation Class Name of the memory
  - *max_len* maximum buffer length

- **getDeviceID**

  Description: Gets the memory Device ID

  Syntax: void (*getDeviceID) (DSDKMemory* pm, char* dev_id, int max_len);

  Parameters:
  - *pm* pointer to memory
  - *dev_id* ID of the memory
  - *max_len* maximum buffer length

- **isVolatile**
  Description: Returns true if the memory is volatile
  Syntax: BOOL (*isVolatile) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: true if the memory is volatile,false otherwise

- **getAccess**
  Description: Gets the memory access type Ex (Read/Write etc).
  Syntax: uint16 (*getAccess) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The memory access.

- **getAccessStr**
  Description: Gets the memory access type as string.
  Syntax: void (*getAccessStr) (DSDKMemory* pm, char* access, int max_len);
  Parameters:
  - *pm* pointer to memory
  - *access* Access type
  - *max_len* maximum buffer length

- **getBlockSize**
  Description: Gets the BlockSize of memory
  Syntax: uint64 (*getBlockSize) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The BlockSize.

- **getNumberOfBlocks**
  Description: Gets the NumberOfBlocks of memory
  Syntax: uint64 (*getNumberOfBlocks) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The Number Of Blocks.

- **getConsumableBlocks**
  Description: Gets the ConsumableBlocks of memory
  Syntax: uint64 (*getConsumableBlocks) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The ConsumableBlocks

- **getEnabledState**
  Description: Gets the EnabledState of the memory
  Syntax: uint16 (*getEnabledState) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The EnabledState

- **getEnabledStateStr**
  Description: Gets the state of the memory as string
  Syntax: void (*getEnabledStateStr) (DSDKMemory* pm, char* state, int max_len);
  Parameters:
  - *pm* pointer to memory
  - *state* The Enabled State is filled in
  - *max_len* maximum buffer length

- **getRequestedState**
  Description: Gets the last RequestedState of the memory
  Syntax: uint16 (*getRequestedState) (DSDKMemory* pm);
  Parameters:
  - *pm* pointer to memory

  Returns: The RequestedState

- **getRequestedStateStr**
  Description: Gets the requested status of the memory as string
  Syntax: void (*getRequestedStateStr) (DSDKMemory* pm, char* state, int max_len);
  Parameters:
  - *pm* pointer to memory
  - *state* Physical memory iterator.
  - *max_len* Physical memory iterator.

- **getOperationalStatus**
  Description: Gets the OperationalStatus
  Syntax: int (*getOperationalStatus) (DSDKMemory* pm, uint16* op_status, int max_types);
  Parameters:
  - *pm* pointer to memory
  - *op_status* Operational Status
  - *max_types* maximum types

- **getOperationalStatusStr**
  Description: Gets the list of operational status as string

Syntax: int (*getOperationalStatusStr) (DSDKMemory* pm, char** os, int max_os, int max_strlen);

Parameters:

- *pm* pointer to memory
- *os* The operational status is filled in.
- *max_os* maximum number of operational status
- *max_strlen* maximum buffer size

- **getHealthState**

  Description: Gets the HealthState of the memory

  Syntax: uint16 (*getHealthState) (DSDKMemory* pm);

  Parameters:

  - *pm* pointer to memory

  Returns: The HealthState

- **getHealthStateStr**

  Description: Gets the health state of the memory as string

  Syntax: void (*getHealthStateStr) (DSDKMemory* pm, char* state, int max_len);

  Parameters:

  - *pm* pointer to memory
  - *state* The health state is filled in.
  - *max_len* maximum buffer length

- **getElementName**

  Description: Gets Element Name of the memory

  Syntax: void (*getElementName) (DSDKMemory* pm, char* ele_name, int max_len);

  Parameters:

  - *pm* pointer to memory
  - *ele_name* ElementName of the memory
  - *max_len* maximum buffer length

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKPhysicalMemoryIterator *di);

  Parameters:

  - *pm* pointer to memory

- **DSDKComputerSystem**

  A structure representing a physical memory

Structure Members

- hdl - Opaque pointer to physical memory specific implementations
- ft - Pointer to physical memory function table.

- **DSDKPhysicalMemoryFT**

  A structure representing physical memory function table.

Member Functions

- getBankLabel
- getCapacity
- getCIMObject
- getDataWidth
- getInterleavePosition
- getPositionInRow
- getSpeed
- getTotalWidth
- getType

Member Functions Description

- **getBankLabel**

  Description**:** Gets the bank label

  Syntax: void(*getBankLabel)(DSDKPhysicalMemory *pm, char *label, int max_len) ;

  Parameters:

  - *pm* Pointer to DSDKPhysicalMemory
  - *label* Pointer to buffer that receives the label
  - *max_len* Maximum length of buffer

- **getCapacity**

  Description**:** Gets the capacity of the memory.

  Syntax: uint64(*getCapacity)(DSDKPhysicalMemory *pm);

  Parameters:

  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The memory capacity.

- **getDataWidth**

  Description**:** Gets the data width

  Syntax: uint16(*getDataWidth)(DSDKPhysicalMemory *pm);

  Parameters:

  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The data width

- **getInterleavePosition**
  Description**:** Gets the interleave position
  Syntax: uint32(*getInterleavePosition)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The interleave position.

- **getPositionInRow**
  Description**:** Gets the position in the row
  Syntax: uint32(*getPositionInRow)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The position in the row

- **getSpeed**
  Description**:** Get the memory speed
  Syntax: uint32(*getSpeed)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The speed.

- **getTotalWidth**
  Description**:** Get the total width.
  Syntax: uint16(*getTotalWidth)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The total width.

- **getTyp**
  Description**:** Get the type of the memory.
  Syntax: uint16(*getType)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

  Returns: The type of the memory.

- **release**
  Description: Releases this object
  Syntax: void(* release)(DSDKPhysicalMemory *pm);
  Parameters:
  - *pm* Pointer to DSDKPhysicalMemory

Note: All functions in C library should call *dsdkc_getLastError* function to get the status

of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- physicalmemory_c.h
Library -- dashapic

5.2.5. PhysicalAsset

- **DSDKPhysicalAssetIterator**

  A structure representing a physical asset iterator.

  Structure Members

  - hdl - Opaque pointer to physical asset specific implementations
  - ft - Pointer to physical asset iterator function table.

- **enumProcessors**

  Description: Enumerate all the physical assets present under a management access point.

  Syntax: DSDKPhysicalAssetIterator* enumPhysicalAssets (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The physical asset iterator.

- **PhysicalAssetIteratorFt**

  A structure representing physical asset iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the physical asset at this iterator location.

  Syntax: DSDKPhysicalAsset*(*getItem)(DSDKPhysicalAssetIterator *di);

  Parameters:

  - *di* Physical asset iterator.

Returns: The physical asset at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKPhysicalAssetIterator *di);

  Parameters:

- *di* Physical asset iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKPhysicalAssetIterator *di);

  Parameters:

- *di* Physical asset iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKPhysicalAssetIterator *di);

  Parameters:

- *di* Physical asset iterator.

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

- **DSDKPhysicalAsset**

  A structure representing a Physical Asset.

  Structure Members

  - hdl - Opaque pointer to physical asset specific implementations
  - ft - Pointer to physical asset function table.

- **DSDKPhysicalAssetFT**

  A structure representing physical asset function table.

Member Functions

- getTag
- getCreationClassName
- getManufacturer
- getModel

- getSerialNumber
- getPartNumber
- getSKU
- getElementName
- canBeFRUed
- getPackageType
- getPackageTypeStr
- getVendorCompatibilityStrings
- getVersion
- getName
- isHostingBoard
- getTypeOfRack
- getTypeOfRackStr
- getChassisPackageType
- getChassisPackageTypeStr
- getConnectorLayout
- getConnectorLayoutStr
- getSlotNumber
- getFormFactor
- getMemoryType
- getMemoryTypeStr
- getMemorySpeed
- getMemoryCapacity
- getMemoryBankLabel
- release

Member Functions Description

- **getTag**

  Description: Get the Tag of the physical asset.

  Syntax: void (*getTag) (DSDKPhysicalAsset* pa, char* tag, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *tag* The tag string
  - *max_len* maximum buffer length

- **getCreationClassName**

  Description: Get the CreationClassName of the physical asset.

  Syntax: void (*getCreationClassName) (DSDKPhysicalAsset* pa, char* name, int max_len);

  Parameters:
  - *pa* pointer to physical asset

- *name* The CreationClassName string
- *max_len* maximum buffer length

- **getManufacturer**

  Description: Get the manufacturer of the physical asset.

  Syntax: void (*getManufacturer) (DSDKPhysicalAsset* pa, char* manufacturer, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *manufacturer* The manufacturer string
  - *max_len* maximum buffer length

- **getModel**

  Description: Gets the model of the physical asset.

  Syntax: void (*getModel) (DSDKPhysicalAsset* pa, char* model, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *model* The model string
  - *max_len* maximum buffer length

- **getSerialNumber**

  Description: Gets the serial number of the physical asset.

  Syntax: void (*getSerialNumber) (DSDKPhysicalAsset* pa, char* serial_number, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *serial_number* The serial number string
  - *max_len* maximum buffer length

- **getPartNumber**

  Description: Gets the part number of the physical asset.

  Syntax: void (*getPartNumber) (DSDKPhysicalAsset* pa, char* part_number, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *part_number* The part number string
  - *max_len* maximum buffer length

- **getSKU**

  Description: Gets the SKU info of the physical asset.

  Syntax: void (*getSKU) (DSDKPhysicalAsset* pa, char* sku, int max_len);

  Parameters:

- *pa* pointer to physical asset
- *sku* The SKU String
- *max_len* maximum buffer length

- **getElementName**

  Description: Gets the Element Name.

  Syntax: void (*getElementName) (DSDKPhysicalAsset* pa, char* name, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *name* The Element Name
  - *max_len* maximum buffer length

- **canBeFRUed**

  Description: Returns true if the physicalasset can be frued

  Syntax: BOOL (*canBeFRUed) (DSDKPhysicalAsset* pa);

  Parameters:
  - *pa* pointer to physical asset

  Returns: true if the physicalasset can be frued,false otherwise

- **getPackageType**

  Description: Gets the Package Type of the physicalasset

  Syntax: uint16 (*getPackageType) (DSDKPhysicalAsset* pa);

  Returns: The Package Type

- **getPackageTypeStr**

  Description: Gets the Package Type of the physicalasset as string

  Syntax: void (*getPackageTypeStr) (DSDKPhysicalAsset* pa, char* type, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *type* The package type
  - *max_len* maximum buffer length

- **getVendorCompatibilityStrings**

  Description: Gets the VendorCompatibilityStrings of the physicalasset

  Syntax: int (*getVendorCompatibilityStrings) (DSDKPhysicalAsset* pa, char**vendor_comp, int max_value, int max_strlen);

  Parameters:
  - *pa* pointer to physical asset
  - *vendor_comp* VendorCompatibilityStrings
  - *max_value* maximum number of values
  - *max_strlen* maximum buffer length

- **getVersion**

  Description: Gets the version of the physical asset.

  Syntax: void (*getVersion) (DSDKPhysicalAsset* pa, char* version, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *version* The version string
  - *max_len* maximum buffer length

- **getName**

  Description: Gets the Name of the physical asset.

  Syntax: void (*getName) (DSDKPhysicalAsset* pa, char* name, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *name* The name string
  - *max_len* maximum buffer length

- **isHostingBoard**

  Description: Returns true if the physicalasset is HostingBoard

  Syntax: BOOL (*isHostingBoard) (DSDKPhysicalAsset* pa);

  Parameters:
  - *pa* pointer to physical asset

  Returns: true if the physicalasset is HostingBoard,false otherwise

- **getTypeOfRack**

  Description: Gets the Type Of Rack of the physicalasset

  Syntax: uint16 (*getTypeOfRack) (DSDKPhysicalAsset* pa);

  Parameters:
  - *pa* pointer to physical asset

  Returns: The Type of Rack

- **getTypeOfRackStr**

  Description: Gets the Type of Rack as string

  Syntax: void (*getTypeOfRackStr) (DSDKPhysicalAsset* pa, char* type, int max_len);

  Parameters:
  - *pa* pointer to physical asset
  - *type* The rack type
  - *max_len* maximum buffer length

- **getChassisPackageType**

  Description: Gets the Chassis Package Type of the physicalasset

Syntax: uint16 (*getChassisPackageType) (DSDKPhysicalAsset* pa);
Parameters:
- pa pointer to physical asset

Returns: The ChassisPackageType

- **getChassisPackageTypeStr**

Description: Gets the chassis Type of the physicalasset as string
Syntax: void (*getChassisPackageTypeStr) (DSDKPhysicalAsset* pa, char* type, int max_len);
Parameters:
- *pa* pointer to physical asset
- *type* The chassi type
- *max_len* maximum buffer length

- **getConnectorLayout**

Description: Gets the Connector Layout of the physicalasset
Syntax: uint16 (*getConnectorLayout) (DSDKPhysicalAsset* pa);
Parameters:
- *pa* pointer to physical asset

Returns: The ConnectorLayout

- **getConnectorLayoutStr**

Description: Gets the connector layout of the physicalasset as string
Syntax: void (*getConnectorLayoutStr) (DSDKPhysicalAsset* pa, char* layout, int max_len);
Parameters:
- *pa* pointer to physical asset
- *layout* The layout
- *max_len* maximum buffer length

- **getSlotNumber**

Description: Gets the Slot Number of the physicalasset
Syntax: uint16 (*getSlotNumber) (DSDKPhysicalAsset* pa);
Parameters:
- *pa* pointer to physical asset

Returns: The SlotNumber

- **getFormFactor**

Description: Gets the Form Factor of the physicalasset
Syntax: uint16 (*getFormFactor) (DSDKPhysicalAsset* pa);
Parameters:
- *pa* pointer to physical asset

Returns: The FormFactor

- **getMemoryType**
  Description: Gets the Memory Type of the physicalasset
  Syntax: uint16 (*getMemoryType) (DSDKPhysicalAsset* pa);
  Parameters:
  - *pa* pointer to physical asset

  Returns: The MemoryType

- **getMemoryTypeStr**
  Description: Gets the Memory Type of the physicalasset as string
  Syntax: void (*getMemoryTypeStr) (DSDKPhysicalAsset* pa, char* type, int max_len);
  Parameters:
  - *pa* pointer to physical asset
  - *type* The memory type
  - *max_len* maximum buffer length

- **getMemorySpeed**
  Description: Gets the Memory Speed of the physicalasset
  Syntax: uint32 (*getMemorySpeed) (DSDKPhysicalAsset* pa);
  Parameters:
  - *pa* pointer to physical asset

  Returns: The MemorySpeed

- **getMemoryCapacity**
  Description: Gets the Memory Capacity of the physicalasset
  Syntax: uint64 (*getMemoryCapacity) (DSDKPhysicalAsset* pa);
  Parameters:
  - *pa* pointer to physical asset

  Returns: The MemoryCapacity

- **getMemoryBankLabel**
  Description: Gets the MemoryBankLabel of the physical asset.
  Syntax: void (*getMemoryBankLabel) (DSDKPhysicalAsset* pa, char* bank_label, int max_len);
  Parameters:
  - *pa* pointer to physical asset
  - *bank_label* The MemoryBankLabel string
  - *max_len* maximum buffer length

- **release**
  Description: Releases this object

Syntax: void(* release)(DSDKPhysicalAssetIterator *di);

Parameters:

- *pa* pointer to physical asset

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- physicalasset_c.h
Library -- dashapic

5.2.6. Fan

- **DSDKFanIterator**

  A structure representing a fan iterator..

  Structure Members

  - hdl - Opaque pointer to fan specific implementations
  - ft - Pointer to fan iterator function table.

- **enumFans**

  Description: Enumerate all the fans present under a management access point.

  Syntax: DSDKFanIterator* enumFans (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Fan iterator.

- **FanIteratorFt**

  A structure representing fan iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the fan at this iterator location.
  Syntax: DSDKFan*(*getItem)(DSDKFanIterator *di);
  Parameters:
  - *di* Fan iterator.

  Returns: The fan at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.
  Syntax: BOOL(* isEnd)(DSDKFanIterator *di);
  Parameters:
  - *di* Fan iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.
  Syntax: void(* next)(DSDKFanIterator *di);
  Parameters:
  - *di* Fan iterator.

- **release**

  Description: Releases this object
  Syntax: void(* release)(DSDKFanIterator *di);
  Parameters:
  - *di* Fan iterator.

- **DSDKFan**

  A structure representing a Fan.
  Structure Members
  - hdl - Opaque pointer to Fan specific implementations
  - ft - Pointer to Fan function table.

- **DSDKFanFT**

  A structure representing fan function table.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getOperationalStatus

- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- isVariableSpeed
- getSpeed
- getDesiredSpeed
- setDesiredSpeed
- isActiveCooling
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getElementName
- release

Member Functions Description

- **getSystemCreationClassName**

  Description: Gets name of the SystemCreationClassName

  Syntax: void (*getSystemCreationClassName) (DSDKFan* fan, char* name, int max_len);

  Parameters:
  - *fan* Pointer to DSDKFan
  - *name* The SystemCreationClassName
  - *max_len* maximum buffer length

- **getSystemName**

  Description: Gets name of the SystemName

  Syntax: void (*getSystemName) (DSDKFan* fan, char* name, int max_len);

  Parameters:
  - *fan* Pointer to DSDKFan
  - *name* The SystemName
  - *max_len* maximum buffer length
- **getCreationClassName**

  Description: Gets name of the CreationClassName

  Syntax: void (*getCreationClassName) (DSDKFan* fan, char* name, int max_len);

  Parameters:
  - *fan* Pointer to DSDKFan
  - *name* The CreationClassName
  - *max_len* maximum buffer length

- **getDeviceID**

Description: Gets name of the Device ID
Syntax: void (*getDeviceID) (DSDKFan* fan, char* devid, int max_len);
Parameters:

- *fan* Pointer to DSDKFan
- *devid* The device ID
- *max_len* maximum buffer length

- **getOperationalStatus**

Description: Gets the OperationalStatus
Syntax: int (*getOperationalStatus) (DSDKFan* fan, uint16* op_status, int max_types);
Parameters:

- *fan* Pointer to DSDKFan
- *op_status* Operational Status
- *max_types* maximum types

- **getOperationalStatusStr**

Description: Gets the list of operational status
Syntax: int (*getOperationalStatusStr) (DSDKFan* fan, char** os, int max_os, int max_strlen);
Parameters:

- *fan* Pointer to DSDKFan
- *os* The operational status is filled in.
- *max_os* maximum number of operational status
- *max_strlen* maximum buffer size

- **getHealthState**

Description: Gets the health state of the fan
Syntax: uint16 (*getHealthState) (DSDKFan* fan);
Parameters:

- *fan* Pointer to DSDKFan

Returns: The health state

- **getHealthStateStr**

Description: Gets the health state of the fan as string
Syntax: void (*getHealthStateStr) (DSDKFan* fan, char* state, int max_len);
Parameters:

- *fan* Pointer to DSDKFan
- *state* The health state is filled in.
- *max_len* maximum buffer length

- **isVariableSpeed**

Description: Returns true if fan supports variable speed. false otherwise

Syntax: BOOL(* DSDKFanFT::isVariableSpeed)(DSDKFan *fan);
Parameters:
- *fan* Pointer to DSDKFan

Return: True if its speed is variable, else false.

- **getSpeed**

Description: Gets the fan's current speed(tach reading).
Syntax: void (*getSpeed) (DSDKFan* fan, char* speed, int max_len);
Parameters:
- *fan* Pointer to DSDKFan
- *speed* The speed of the fan.
- *max_len* maximum buffer length

- **getDesiredSpeed**

Description**:** Gets the desired speed of the fan
Syntax: uint64(* DSDKFanFT::getDesiredSpeed)(DSDKFan *fan);
Parameters:
- *fan* Pointer to DSDKFan

Returns: The desired speed of the fan.

- **setDesiredSpeed**

Description: Sets the desired speed of the fan
Syntax: BOOL(* DSDKFanFT::setDesiredSpeed)(DSDKFan *fan,
uint64 speed)
Parameters:
- *speed* The desired speed.

Returns: True if the speed is set, false if unable to set the speed.

- **isActiveCooling**

Description: Returns true if the fan supports isActiveCooling
Syntax: BOOL (*isActiveCooling) (DSDKFan* fan);
Parameters:
- *fan* Pointer to DSDKFan

Returns: true if fan supports variable speed,false otherwise

- **getEnabledState**

Description: Gets the enabled state of the fan
Syntax: uint16 (*getEnabledState) (DSDKFan* fan);
Parameters:
- *fan* Pointer to DSDKFan

Returns: The enabled state of the fan.

- **getEnabledStateStr**

  Description: Gets the state of the fan as string

  Syntax: void (*getEnabledStateStr) (DSDKFan* fan, char* state, int max_len);

  Parameters:

  - *fan* Pointer to DSDKFan
  - *state* The Enabled State is filled in
  - *max_len* maximum buffer length

- **getRequestedState**

  Description: Gets the requested state of the fan

  Syntax: uint16 (*getRequestedState) (DSDKFan* fan);

  Returns: The requested state of the fan.

- **getRequestedStateStr**

  Description: Gets the requested status of the fan as string

  Syntax: void (*getRequestedStateStr) (DSDKFan* fan, char* state, int max_len);

  Parameters:

  - *fan* Pointer to DSDKFan
  - *state* The element name
  - *max_len* maximum buffer length

- **getElementName**

  Description: Gets the name of the Element

  Syntax: void (*getElementName) (DSDKFan* fan, char* ele_name, int max_len);

  Parameters:

  - *fan* Pointer to DSDKFan
  - *ele_name* The element name
  - *max_len* maximum buffer length

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKFan* fan);

  Parameters:

  - *fan* Pointer to DSDKFan

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- fan_c.h
Library -- dashapic

5.2.7. Sensor

- **DSDKSensorIterator**
  A structure representing a sensor iterator..

  Structure Members
  - hdl- Opaque pointer to sensor specific implementations
  - ft - Pointer to sensor iterator function table.

- **enumProcessors**
  Description: Enumerate all the sensor present under a management access point.
  Syntax: DSDKSensorIterator* enumSensors (DSDKClient* client, BOOL cached);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Sensor iterator.

- **SensorIteratorFt**
  A structure representing sensor iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**
  Description: Gets the sensor at this iterator location.
  Syntax: DSDKSensor*(*getItem)(DSDKSensorIterator *di);
  Parameters:
  - *di* Processor iterator.

  Returns: The sensor at this iterator location.
- **isEnd**
  Description: Returns true if iterator have reached the end.
  Syntax: BOOL(* isEnd)(DSDKSensorIterator *di);

Parameters:

- *di* Sensor iterator.

Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKSensorIterator *di);

  Parameters:

  - *di* Sensor iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKSensorIterator *di);

  Parameters:

  - *di* Sensor iterator.

- **DSDKSensor**

  A structure representing a Sensor

  Structure Members

  - hdl - Opaque pointer to sensor specific implementations
  - ft - Pointer to sensor function table.

- **DSDKSensorFT**

  A structure representing sensor function table.

Member Functions

- **getSystemCreationClassName**
- getSystemName
- getCreationClassName
- getDeviceID
- getSensorType
- getSensorTypeStr
- getPossibleStates
- getCurrentState
- getElementName
- getOtherSensorTypeDescription
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getOperationalStatus

- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- release

Member Functions Description

- **getSystemCreationClassName**

  Description: Gets name of the SystemCreationClassName

  Syntax: void (*getSystemCreationClassName) (DSDKSensor* sensor, char* name, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *name* The SystemCreationClassName
  - *max_len* maximum buffer length

  Returns: none

- **getSystemName**

  Description: Gets the SystemName

  Syntax: void (*getSystemName) (DSDKSensor* sensor, char* name, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *name* The SystemName
  - *max_len* maximum buffer length

  Returns: none

- **getCreationClassName**

  Description: Gets the CreationClassName

  Syntax: void (*getCreationClassName) (DSDKSensor* sensor, char* name, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *name* The CreationClassName
  - *max_len* maximum buffer length

  Returns: none

- **getDeviceID**

  Description: Gets the Device ID

  Syntax: void (*getDeviceID) (DSDKSensor* sensor, char* devid, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor

- *devid* The device ID
- *max_len* maximum buffer length

Returns none

- **getSensorType**

  Description: Gets the type of the sensor

  Syntax: uint16 (*getSensorType) (DSDKSensor* sensor);

  Parameters:
  - *sensor* Pointer to DSDKSensor

  Returns: The sensor type.

- **getSensorTypeStr**

  Description: Gets the type of the sensor as string

  Syntax: void (*getSensorTypeStr) (DSDKSensor* sensor, char* sensor_type, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *sensor_type* The sensor Type string
  - *max_len* maximum buffer length

  returns: none

- **getPossibleStates**

  Description: Gets the possible states of this sensor.

  Syntax: int (*getPossibleStates) (DSDKSensor* sensor, char** possible_state, int max_state, int max_strlen);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *possible_state* The possible states of this sensor.
  - *max_state* maximum number of states
  - *max_strlen* maximum buffer size

- **getCurrentState**

  Description: Gets the current state of this sensor.

  Syntax: void (*getCurrentState) (DSDKSensor* sensor, char* state, int max_len);

  Parameters:
  - *sensor* Pointer to DSDKSensor
  - *state* The current state of this sensor.
  - *max_len* maximum buffer length

  Returns: none

- **getElementName**

  Description: Gets the name of the Element

Syntax: void (*getElementName) (DSDKSensor* sensor, char* ele_name, int max_len);

Parameters:

- *sensor* Pointer to DSDKSensor
- *ele_name* The element name
- *max_len* maximum buffer length

Returns: none

- **getOtherSensorTypeDescription**

  Description: Gets the sensor type description if the sensor type is "other".

  Syntax: void (*getOtherSensorTypeDescription) (DSDKSensor* sensor, char* type_desc, int max_len);

  Parameters:

  - *sensor* Pointer to DSDKSensor
  - *type_desc* The sensor type description
  - *max_len* maximum buffer length

  Returns: none

- **getEnabledState**

  Description: Gets the enabled state of the sensor

  Syntax: uint16 (*getEnabledState) (DSDKSensor* sensor);

  Parameters:

  - *sensor* Pointer to DSDKSensor

  Returns: The enabled state of the sensor.

- **getEnabledStateStr**

  Description: Gets the state of the sensor as string

  Syntax: void (*getEnabledStateStr) (DSDKSensor* sensor, char* state, int max_len);

  Parameters:

  - *sensor* Pointer to DSDKSensor
  - *state* The Enabled State is filled in
  - *max_len* maximum buffer length

  Returns: none

- **getRequestedState**

  Description: Gets the requested state of the sensor

  Syntax: uint16 (*getRequestedState) (DSDKSensor* sensor);

  Parameters:

  - *sensor* Pointer to DSDKSensor

  Returns: The requested state of the sensor.

- **getRequestedStateStr**

Description: Gets the requested status of the sensor as string
Syntax: void (*getRequestedStateStr) (DSDKSensor* sensor, char* state, int max_len);
Parameters:
- *sensor* Pointer to DSDKSensor
- *state* The requested state is filled in.
- *max_len* maximum buffer length

Returns: none

- **getOperationalStatus**

Description: Gets the OperationalStatus
Syntax: int (*getOperationalStatus) (DSDKSensor* sensor, uint16* op_status, int max_types);
Parameters:
- *sensor* Pointer to DSDKSensor
- *op_status* Operational Status
- *max_types* maximum types

- **getOperationalStatusStr**

Description: Gets the list of operational status
Syntax: int (*getOperationalStatusStr) (DSDKSensor* sensor, char** os, int max_os, int max_strlen);
Parameters:
- *sensor* Pointer to DSDKSensor
- *os* The operational status is filled in.
- *max_os* maximum number of operational status
- *max_strlen* maximum buffer size

- **getHealthState**

Description: Gets the health state of the sensor
Syntax: uint16 (*getHealthState) (DSDKSensor* sensor);
Parameters:
- *sensor* Pointer to DSDKSensor

returns: The health state

- **getHealthStateStr**

Description: Gets the health state of the sensor as string
Syntax: void (*getHealthStateStr) (DSDKSensor* sensor, char* state, int max_len);
Parameters:
- *sensor* Pointer to DSDKSensor
- *state* The health state is filled in.

- *max_len* maximum buffer length

Returns: none

- **release**

Description: Releases this object

Syntax: void (*release) (DSDKSensor* sensor);

Parameters:

- sensor Pointer to DSDKSensor

Returns: **none**

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file sensor_c.h

Library dashapic

5.2.8. Software

- **DSDKSoftwareIterator**

A structure representing a software iterator.

Structure Members

- hdl - Opaque pointer to software specific implementations
- ft - Pointer to software iterator function table.

- **enumSoftware**

Description: Enumerate all the software present under a management access point.

Syntax: DSDKSoftwareIterator* enumSoftware (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The Software iterator.

- **SoftwareIteratorFt**

A structure representing software iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**
  Description: Gets the software at this iterator location.
  Syntax: DSDKSoftware*(*getItem)(DSDKSoftwareIterator *di);
  Parameters:
  - *di* Software iterator.

  Returns: The software at this iterator location.

- **isEnd**
  Description: Returns true if iterator have reached the end.
  Syntax: BOOL(*isEnd)(DSDKSoftwareIterator *di);
  Parameters:
  - *di* Software iterator.

  Returns: True if have reached the end.

- **next**
  Description: Moves the iterator to the next location.
  Syntax: void(*next)(DSDKSoftwareIterator *di);
  Parameters:
  - *di* Software iterator.

- **release**
  Description: Releases this object
  Syntax: void(* release)(DSDKSoftwareIterator *di);
  Parameters:
  - *di* Software iterator.

- **DSDKSoftware**
  A structure representing a Software.
  Structure Members
  - hdl - Opaque pointer to Software specific implementations
  - ft - Pointer to Software function table.

- **DSDKSoftwareFT**
  A structure representing software function table.

Member Functions

- getInstanceID
- getIsEntity
- getVersionString
- getMajorVersion
- getMinorVersion
- getRevisionNumber
- getBuildNumber
- getTargetOSTypes
- getTargetOperatingSystems
- getIdentityInfoType
- getIdentityInfoValue
- getClassifications
- getClassificationsStr
- release

Member Functions Description

- **getInstanceID**

  Descriptions: Gets the Instance ID of software

  Syntax: void (*getInstanceID) (DSDKSoftware* sw, char* instanceid, int max_len);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *instanceid* Instance ID of software
  - *max_len* maximum buffer length

  Returns: none

- **getIsEntity**

  Descriptions: Gets IsEntity - whether the SoftwareIdentity corresponds to a discrete copy of the software component or is being used to convey descriptive and identifying information about software that is not present in the management domain.A value of TRUE shall indicate that the SoftwareIdentity instance corresponds to a discrete copy of the software component. A value of FALSE shall indicate that the SoftwareIdentity Instance does not correspond to a discrete copy of the Software.

  Syntax: BOOL (*getIsEntity) (DSDKSoftware* sw);

  Parameters:

  - *sw* Pointer to DSDKSoftware

  Returns: The boolean value true if it is Entity else false

- **getVersionString**

  Desciptions: Gets the version string

  Syntax: void (*getVersionString) (DSDKSoftware* sw, char* version, int max_len);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *version* buffer to store version
  - *max_len* maximum buffer length

  Returns none

- **getMajorVersion**

  Description: Gets the major version

  Syntax: uint16 (*getMajorVersion) (DSDKSoftware* sw);

  Parameters:

  - *sw* Pointer to DSDKSoftware

  Returns: The major version

- **getMinorVersion**

  Description: Gets the minor version

  Syntax: uint16 (*getMinorVersion) (DSDKSoftware* sw);

  Parameters:

  - *sw* Pointer to DSDKSoftware

  Returns: The minor version

- **getRevisionNumber**

  Description: Gets the revision number

  Syntax: uint16 (*getRevisionNumber) (DSDKSoftware* sw);

  Parameters:

  - *sw* Pointer to DSDKSoftware

  Returns: The revision number

- **getBuildNumber**

  Description: Gets the build number

  Syntax: uint16 (*getBuildNumber) (DSDKSoftware* sw);

  Parameters:

  - *sw* Pointer to DSDKSoftware

  Returns: The build number

- **getTargetOSTypes**

  Description: Gets the list of target operating systems types

  Syntax: int (*getTargetOSTypes) (DSDKSoftware* sw, uint16* ostypes, int max_os_types);

Parameters:

- *sw* Pointer to DSDKSoftware
- *ostypes* The target operating systems types
- *max_os_types* maximum number of operating systems types

- **getTargetOperatingSystems**

  Description: Gets the list of target operating systems

  Syntax: int (*getTargetOperatingSystems) (DSDKSoftware* sw, char** os, int max_os, int max_strlen);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *os* The target operating systems
  - *max_os* maximum number of operating systems
  - *max_strlen* maximum buffer size

- **getIdentityInfoType**

  Description: Gets the list of IdentityInfoType

  Syntax: int (*getIdentityInfoType) (DSDKSoftware* sw, char** identityinfotype, int max_info_type, int max_strlen);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *identityinfotype* The IdentityInfoType
  - *max_info_type* maximum number of IdentityInfoType
  - *max_strlen* maximum buffer size

- **getIdentityInfoValue**

  Description: Gets the list of IdentityInfoValue

  Syntax: int (*getIdentityInfoValue) (DSDKSoftware* sw, char** identityInfoValue, int max_info_value, int max_strlen);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *identityInfoValue* The IdentityInfoValue
  - *max_info_value* maximum number of IdentityInfoType
  - *max_strlen* maximum buffer size

- **getClassifications**

  Descripition: Gets the classification

  Syntax: int (*getClassifications) (DSDKSoftware* sw, uint16* classification, int max_classification);

  Parameters:

  - *sw* Pointer to DSDKSoftware
  - *classification* software classification
  - *max_classification* maximum classifications

- **getClassificationsStr**

  Description: Gets the classification as string

  Syntax: int (*getClassificationsStr) (DSDKSoftware* sw, char** classification, int max_classification, int max_strlen);

  Parameters:
  - *sw* Pointer to DSDKSoftware
  - *classification* software classification
  - *max_classification* maximum classifications

- **release**

  Description: Releases this object

  Syntax: void (*release) (DSDKSoftware* sw);

  Parameters:
  - *sw* Pointer to DSDKSoftware

  returns: none

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- software_c.h
Library -- dashapic

5.2.9. BootConfig

- **DSDKBootConfigIterator**

  A structure representing a processor iterator.

  Structure Members
  - hdl - Opaque pointer to processor specific implementations
  - ft - Pointer to processor iterator function table.

- **enumBootConfigs**

  Description: Enumerate all the boot configurations present under a management access point.

  Syntax: DSDKBootConfigIterator* enumBootConfigs (DSDKClient* client, BOOL cached);

  Parameters:
  - *client* Pointer to the client interface.

- *cached* Enable/Disable caching.

Returns: The Boot Configuration iterator.

- **BootConfigIteratorFt**
  A structure representing boot configuration iterator function table.

Member Functions

- getInstanceID
- getElementName
- getBootString
- getBIOSBootString
- getStructuredBootString
- getFailThroughSupported
- release

Member Functions Description

- **getInstanceID**

  Description: Gets theInstanceID

  Syntax: void (*getInstanceID) (DSDKBootDevice* bd, char* ins_id, int max_len);

  Parameters:
  - *bd* Pointer to DSDKBootDevice
  - *ins_id* buffer to store instance ID
  - *max_len* MAximum buffer length

  Returns none

- **getElementName**

  Description: Gets the element name

  Syntax: void (*getElementName) (DSDKBootDevice* bd, char* element_name, int max_len);

  Parameters:
  - *bd* Pointer to DSDKBootDevice
  - *element_name* Bootconfig element name
  - *max_len* maximum buffer length

  Returns none

- **getBootString**

  Description: Gets the BootString

  Syntax: void (*getBootString) (DSDKBootDevice* bd, char* bootstring, int max_len);

  Parameters:
  - *bd* Pointer to DSDKBootDevice

- *bootstring* BootString
- *max_len* maximum buffer length

Returns: none

- **getBIOSBootString**

  Description: Gets the BIOSBootString

  Syntax: void (*getBIOSBootString) (DSDKBootDevice* bd, char* biosbootstring, int max_len);

  Parameters:
  - *bd* Pointer to DSDKBootDevice
  - *biosbootstring* buffer to store BIOSBootString
  - *max_len* maximum buffer length

  Returns none

- **getStructuredBootString**

  Description: Gets the StructuredBootString

  syntax: void (*getStructuredBootString) (DSDKBootDevice* bd, char* str_bootstring, int max_len);

  Parameters:
  - *bd* Pointer to DSDKBootDevice
  - *str_bootstring* StructuredBootString
  - *max_len* maximum buffer length

  Returns none

- **getFailThroughSupported**

  Description: Gets fail through supported

  Syntax: uint16 (*getFailThroughSupported) (DSDKBootDevice* bd);

  Parameters:
  - *bd* Pointer to DSDKBootDevice

  Returns: The fail through supported

- **release**

  Description: Releases this object

  Syntax: void (*release) (DSDKBootDevice* bd);

  Parameters:
  - *bd* Pointer to DSDKBootDevice

  Returns: none

- **DSDKBootConfig**

  A structure representing a Boot Configuration

Structure Members

- hdl - Opaque pointer to boot configuration specific implementations
- ft - Pointer to Boot configuration function table.

- **DSDKBootConfigFT**
  A structure representing boot configuration function table.

Member Functions

- getInstanceID
- getElementName
- isDefaultBoot
- isCurrentBoot
- isNextBoot
- setDefaultBoot
- setNextBoot
- getBootOrder
- changeBootOrder
- addBootConfig
- deleteBootConfig
- release

Member Function Description

- **getInstanceID**
  Description: Gets theInstanceID
  Syntax: void (*getInstanceID) (DSDKBootConfig* bc, char* ins_id, int max_len);
  Parameters:
  - *bc* Pointer to DSDKBootConfig
  - *ins_id* InstanceID
  - *max_len* maximum buffer length

  Returns: None

- **getElementName**
  Description: Gets the element name
  Syntax: void (*getElementName) (DSDKBootConfig* bc, char* element_name, int max_len);
  Parameters:
  - *bc* Pointer to DSDKBootConfig
  - *element_name* Bootconfig element name
  - *max_len* maximum buffer length

  Returns None

- **isDefaultBoot**

Description: Is this default boot configuration
Syntax: BOOL (*isDefaultBoot) (DSDKBootConfig* bc);
Parameters:
- *bc* Pointer to DSDKBootConfig

Returns true if success false otherwise

- **isCurrentBoot**

Description: Is this current boot configuration
Syntax: BOOL (*isCurrentBoot) (DSDKBootConfig* bc);
Parameters:
- *bc* Pointer to DSDKBootConfig

Returns true if success false otherwise

- **isNextBoot**

Description: Is this next boot configuration
Syntax: BOOL (*isNextBoot) (DSDKBootConfig* bc);
Parameters:
- *bc* Pointer to DSDKBootConfig

Returns true if success false otherwise

- **setDefaultBoot**

Description: Set this configuration as default configuration
Syntax: void (*setDefaultBoot) (DSDKBootConfig* bc);
BOOL (*isNextBoot) (DSDKBootConfig* bc);
Parameters:
- *bc* Pointer to DSDKBootConfig

Returns None

- **setNextBoot**

Description: Set this configuration next boot.
Syntax: void (*setNextBoot) (DSDKBootConfig* bc);
Parameters:
- *bc* Pointer to DSDKBootConfig

Returns None

- **getBootOrder**

Description: Gets the boot order
Syntax: int (*getBootOrder) (DSDKBootConfig* bc, DSDKBootDevice**
boot_order,Int max_device);
Parameters:
- *bc* Pointer to DSDKBootConfig
- *boot_order* Boot order list

- *max_device* maximum devices

Returns boot order

- **changeBootOrder**

  Description: Changes boot order

  Syntax: void (*changeBootOrder) (DSDKBootConfig* bc, DSDKBootDevice* boot_order[], int num_device);

  Parameters:

  - *bc* Pointer to DSDKBootConfig
  - *boot_order* changed boot order
  - *num_device* maximum buffer length

- **addBootConfig**

  Description: Adds Boot Config

  Syntax: DSDKBootConfig* (*addBootConfig) (DSDKComputerSystem* cs);

  Parameters:

  - cs Pointer to computersystem to add bootconfig

  Returns None

- **deleteBootConfig**

  Description: Delete this boot config

  Syntax: void (*deleteBootConfig) (DSDKBootConfig* bc);

  Parameters:

  - *bc* Pointer to DSDKBootConfig

  returns none

- **release**

  Description: Releases this object

  Syntax: void (*release) (DSDKBootConfig* bc);

  Parameters:

  - *bc* Pointer to DSDKBootConfig

  returns none

Note: All functions in C library should call dsdkc_getLastError function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use dsdkc_getLastErrorStr to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- software_c.h

Library -- dashapic

5.2.10. User

- **DSDKUserIterator**

  A structure representing a user iterator.

  Structure Members

  - hdl - Opaque pointer to user specific implementations
  - ft - Pointer to user iterator function table.

- **enumUsers**

  Description: Enumerate all the users present under a management access point.

  Syntax: DSDKUserIterator* enumUsers (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *Cached* Enable/Disable caching.

  Returns: The User iterator.

- **UserIteratorFt**

  A structure representing user iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the processor at this iterator location.

  Syntax: DSDKUser*(*getItem)(DSDKUserIterator *di);

  Parameters:

  - *di* User iterator.

  Returns: The user at this iterator location.

- **isEnd**

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKUserIterator *di);
Parameters:
- *di* User iterator.

Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.
  Syntax: void(* next)(DSDKUserIterator *di);
  Parameters:
  - *di* User iterator.

- **release**

  Description: Releases this object
  Syntax: void(* release)(DSDKUserIterator *di);
  Parameters:
  - *di* User iterator.

- **DSDKUser**

  A structure representing a User

  Structure Members

  - hdl - Opaque pointer to User specific implementations
  - ft - Pointer to user function table.

- **DSDKUserFT**

  A structure representing user function table.

Member Functions

- createUser
- getUserRoles
- assignRoles
- removeRoles
- deleteUser
- enableUser
- disableUser
- getSystemCreationClassName
- getCreationClassName
- getSystemName
- getName
- getUserID

- getUserPassword
- getOrganizationName
- getElementName
- getUserPasswordEncryptionAlgorithm
- OtherUserPasswordEncryptionAlgorithm
- getPasswordHistoryDepth
- getComplexPasswordRulesEnforced
- getMaximumSuccessiveLoginFailures
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- release

Member Functions Description

- **createUser**

  Description: Creats a user

  Syntax: DSDKUser* (*createUser) (DSDKComputerSystem* cs,
  char*user_name, char*password,char*certificate);

  Parameters:
  - *cs* Pointer to DSDKComputerSystem
  - *user_name* Pointer to buffer that receives the user name
  - *password* Pointer to buffer that receives the password
  - *certificate* Pointer to buffer that receives the certificate

- **getUserRoles**

  Description: Gets the associated role with this user

  Syntax: int (*getUserRoles) (DSDKUser* user, DSDKRole** role);

  Parameters:
  - *user* Pointer to DSDKUser
  - *role* Role names to get for this user.

- **assignRoles**

  Description: Assign Role(s) to this user.

  Syntax: void (*assignRoles) (DSDKUser* user,char** roles,int num_values);

  Parameters:
  - *user* Pointer to DSDKUser
  - *roles* Role names to assign for this user.

- **removeRoles**

  Description: Remove Role(s) from this user

  Syntax: void (*removeRoles) (DSDKUser* user, char** roles,int num_values);

Parameters:

- *user* Pointer to DSDKUser
- *roles* Role names to remove for this user.

- **deleteUser**

  Description: Delets this user

  Syntax: void (*deleteUser) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

- **enableUser**

  Description: EnableUser

  Syntax: uint32 (*enableUser) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

- **disableUser**

  Description: DisableUser

  Syntax: uint32 (*disableUser) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

- **getSystemCreationClassName**

  Description: Gets SystemCreationClassName

  Syntax: void (*getSystemCreationClassName) (DSDKUser* user, char* str, int max_len);

  Parameters:

  - *user* Pointer to DSDKUser
  - *str* Pointer to buffer that receives the name
  - *max_len* Maximum length of buffer

  Returns : The systemcreationclassname

- **getCreationClassName**

  Description: Gets CreationClassName

  Syntax: void (*getCreationClassName) (DSDKUser* user, char* str, int max_len);

  Parameters:

  - *user* Pointer to DSDKUser
  - *str* Pointer to buffer that receives the name
  - *max_len* Maximum length of buffer

  Returns : The CreationClassName

- **getSystemName**

Description: Gets SystemName
Syntax: void (*getSystemName) (DSDKUser* user, char* str, int max_len);
Parameters:

- *user* Pointer to DSDKUser
- *str* Pointer to buffer that receives the name
- *max_len* Maximum length of buffer

Returns : The SystemName

- **getName**

Description: Gets the user name
Syntax: void(*getName)(DSDKUser *user, char *name, int max_len);
Parameters:

- *user* Pointer to DSDKUser
- *name* Pointer to buffer that receives the name
- *max_len* Maximum length of buffer

Returns : the user name

- **getUserID**

Description: Gets user id for this user
Syntax: void (*getUserID) (DSDKUser* user, char* userid, int max_len);
Parameters:

- *user* Pointer to DSDKUser
- *userid* Pointer to buffer that receives the user id
- *max_len* Maximum length of buffer

Returns : The user id

- **getUserPassword**

Description: Gets an array of UserPassword
Syntax: int (*getUserPassword) (DSDKUser* user, char** users, int max_users, int max_strlen);
Parameters:

- *user* Pointer to DSDKUser
- *users* Pointer to buffer that receives the users
- *max_users* Maximum number of users
- max_strlen Maximum length of buffer

Returns : The user password

- **getOrganizationName**

Description: Gets an array of OrganizationName
Syntax: int (*getOrganizationName) (DSDKUser* user, char** orgname, int max_orgname, int max_strlen);
Parameters:

- *user* Pointer to DSDKUser
- *orgname* Pointer to buffer that receives the org name
- max_orgname Maximum length of buffer

Returns : The organizationname

- **getElementName**

Description: Gets ElementName

Syntax: void (*getElementName) (DSDKUser* user, char* name, int max_len);

Parameters:

- *user* Pointer to DSDKUser
- *name* Pointer to buffer that receives the name
- *max_len* Maximum length of buffer

Returns : The ElementName

- **getUserPasswordEncryptionAlgorithm**

Description: Gets the UserPasswordEncryptionAlgorithm

Syntax: uint16 (*getUserPasswordEncryptionAlgorithm)(DSDKUser* user);

Parameters:

- *user* Pointer to DSDKUser

- **OtherUserPasswordEncryptionAlgorithm**

Description: Gets OtherUserPasswordEncryptionAlgorithm

Syntax: void (*OtherUserPasswordEncryptionAlgorithm) (DSDKUser* user, char* name, int max_len);

Parameters:

- *user* Pointer to DSDKUser
- *name* Pointer to buffer that receives the name
- *max_len* Maximum length of buffer

Returns : The otheruserpasswordencryptionalgorithm

- **getPasswordHistoryDepth**

Description: Gets PasswordHistoryDepth

Syntax: uint16 (*getPasswordHistoryDepth) (DSDKUser* user);

Parameters:

- *user* Pointer to DSDKUser
- *name* Pointer to buffer that receives the name
- *max_len* Maximum length of buffer

Returns : PasswordHistoryDepth

- **getComplexPasswordRulesEnforced**

Description: Gets an array of ComplexPasswordRulesEnforced

Syntax: int (*getComplexPasswordRulesEnforced) (DSDKUser* user, uint16*

rules, int max_rules);

Parameters:

- *user* Pointer to DSDKUser
- *rules* Pointer to buffer that receives the rules
- *max_rules* Maximum length of buffer

Returns : The list of complexpasswordrulesenforced

- **getMaximumSuccessiveLoginFailures**

  Description: Gets MaximumSuccessiveLoginFailures

  Syntax: uint16 (*getMaximumSuccessiveLoginFailures) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

  Returns : The MaximumSuccessiveLoginFailures

- **getEnabledState**

  Description: Gets the EnabledState of the user

  Syntax: uint16 (*getEnabledState) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

  Returns : The EnabledState of the user

- **getEnabledStateStr**

  Description: Gets the state of the user as string

  Syntax: void (*getEnabledStateStr) (DSDKUser* user, char* state, int max_len);

  Parameters:

  - *user* Pointer to DSDKUser
  - *state* Pointer to buffer that receives the state
  - *max_len* Maximum length of buffer

  Returns : The EnabledState of the user as string

- **getRequestedState**

  Description: Gets the last RequestedState of the user

  Syntax: uint16 (*getRequestedState) (DSDKUser* user);

  Parameters:

  - *user* Pointer to DSDKUser

  Returns : The last RequestedState of the user

- **getRequestedStateStr**

  Description: Gets the last RequestedState of the user as string

  Syntax: void (*getRequestedStateStr) (DSDKUser* user, char* state, int max_len);

  Parameters:

- *user* Pointer to DSDKUser
- *state* Pointer to buffer that receives the state
- *max_len* Maximum length of buffer

Returns : The last RequestedState of the user as string

- **release**

  Description: Releases this object

  Syntax: void(*release)(DSDKUser *user);

  Parameters:
  - *user* Pointer to DSDKUser

- **DSDKRoleIterator**

  A structure representing a role iterator.

  Structure Members
  - hdl - Opaque pointer to role specific implementations
  - ft - Pointer to role iterator function table.

- **enumRoles**

  Description: Enumerate all the roles present under a management access point.

  Syntax: DSDKRoleIterator* enumRoles (DSDKClient* client, BOOL cached);

  Parameters:
  - *client* Pointer to the client interface.
  - *Cached* Enable/Disable caching.

  Returns: The Role iterator.

- **RoleIteratorFt**

  A structure representing role iterator function table.

Member Functions
- getItem
- isEnd
- next
- release

Member Functions Description

- **getItem**

  Description: Gets the processor at this iterator location.

  Syntax: DSDKRole*(*getItem)(DSDKRoleIterator *di);

Parameters:

- *di* Role iterator.

Returns: The role at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(* isEnd)(DSDKRoleIterator *di);

  Parameters:

  - *di* Role iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(* next)(DSDKRoleIterator *di);

  Parameters:

  - *di* Role iterator.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKRoleIterator *di);

  Parameters:

  - *di* Role iterator.

- **DSDKRole**

  A structure representing a Role

  Structure Members

  - hdl - Opaque pointer to Role specific implementations
  - ft - Pointer to role function table.

- **DSDKRoleFT**

  A structure representing role function table.

- **DSDKRoleIterator**

  A structure representing a role iterator.

  Structure Members

  - hdl - Opaque pointer to user specific implementations
  - ft - Pointer to user iterator function table.

- **enumRoles**

  Description: Enumerate all the users present under a management access

point.

Syntax: DSDKRoleIterator* enumRoles (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client interface.
- *Cached* Enable/Disable caching.

Returns: The Role iterator.

- **RoleIteratorFt**
  A structure representing role iterator function table.

Member Functions

- getCreationClassName
- getName
- getRoleCharacteristics
- getCommonName
- getElementName
- createRole
- deleteRole
- modifyRole
- assignPermissions
- getPermissions
- getSupportedActivityQualifiers
- release

Member Function Description:

- **getCreationClassName**
  Description: Gets the role's Creation class Name
  Syntax: void (*getCreationClassName) (DSDKRole* role, char* name, int max_len);
  Parameters:
  - *role* Pointer to DSDKRole
  - *name* Buffer to store creation class name
  - *max_len* maximum buffer length

  Returns : none

- **getName**
  Description: Gets the name of this Role
  Syntax: void (*getName) (DSDKRole* role, char* name, int max_len);
  Parameters:

- *role* Pointer to DSDKRole
- *name* Buffer to store role name
- *max_len* maximum buffer length

Returns : none

- **getRoleCharacteristics**

  Description: Gets the RoleCharacteristics

  Syntax: int (*getRoleCharacteristics) (DSDKRole* role, uint16* role_char, int max_types);

  Parameters:
  - *role* Pointer to DSDKRole
  - *role_char* RoleCharacteristics
  - *max_len* maximum buffer length

- **getCommonName**

  Description: Gets the CommonName of this Role

  Syntax: void (*getCommonName) (DSDKRole* role, char* commonname, int max_len);

  Parameters:
  - *role* Pointer to DSDKRole
  - *name* The CommonName
  - *max_len* maximum buffer length

- **getElementName**

  Description: Gets the ElementName of this Role

  Syntax: void (*getElementName) (DSDKRole* role, char* name, int max_len);

  Parameters:
  - *role* Pointer to DSDKRole
  - *name* The ElementName
  - *max_len* maximum buffer length

- **createRole**

  Description: Creats a new role with the permissions specified

  Syntax: DSDKRole* (*createRole) (DSDKComputerSystem* cs, char*name, RolePermission_T* permissions, int num_permission);

  Parameters:
  - *cs* Computer System where the role will be added.
  - *name* Name of the role
  - *permissions* Privileges/permissions for this role(Permission_T (activities, qualifiers & formats))

    Permissions are created using the

1. activities (Activity_E) (Create|Delete|Detect|Read|Write| Execute|Other).
2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
3. and formats (QualifierFormats_E), this is optional(targets may or may not support this).

- **deleteRole**

  Description: Deletes this role

  Syntax: void (*deleteRole) (DSDKRole* role);

  Parameters:

  - *role* Pointer to DSDKRole

- **modifyRole**

  Description: Modify this role with new privileges

  Syntax: void (*modifyRole) (DSDKRole* role, RolePermission_T* permissions, int num_permission);

  Parameters:

  - *role* Pointer to DSDKRole
  - *permissions* Privileges/permissions for this role(Permission_T (activities, qualifiers & formats))

    Permissions are created using the
    1. activities (Activity_E) (Create|Delete|Detect|Read|Write| Execute|Other).
    2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
    3. and formats (QualifierFormats_E), this is optional(targets may or may not support this).

- **assignPermissions**

  Description: Gets the permissions for this role

  Syntax: int (*getPermissions) (DSDKRole* role, RolePermission_T* permissions,int max_permission);

  Parameters:

  - *role* Pointer to DSDKRole
  - *permissions* Privileges/permissions for this role(Permission_T (activities, qualifiers & formats))

    Permissions are created using the
    1. activities (Activity_E) (Create|Delete|Detect|Read|Write| Execute|Other).
    2. qualifiers (valid qualifiers are obtained using getSupportedActivityQualifiers),
    3. and formats (QualifierFormats_E), this is optional(targets may or may not support this).

- **getSupportedActivityQualifiers**

  Description: Gets the activity qualifiers supported by the target/MAP.

  Syntax: int (*getSupportedActivityQualifiers) (DSDKRole* role, DSDKClient* client, char** activityqualifiers, int max_qualifiers, int max_strlen);

  Parameters:

  - *role* Pointer to DSDKRole
  - client Pointer to client iterator
  - activityqualifiers buffer to store activity qialifier
  - max_qualifier maximun number of qualifiers
  - max_strlen maximum buffer length

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- user_c.h
Library -- dashapic

5.2.11. PowerSupply

- **DSDKPowerSupplyIterator**

  A structure representing a power supply iterator..

  Structure Members

  - hdl - Opaque pointer to power supply specific implementations
  - ft - Pointer to power supply iterator function table.

- **enumPowerSupplies**

  Description: Enumerate all the power supplies present under a management access point.

  Syntax: DSDKPowerSupplyIterator* enumPowerSupplies (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Power supply iterator.

- **PowerSupplyIteratorFt**

A structure representing power supply iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Function Descriptions

- **getItem**
  Description: Gets the power supply at this iterator location.
  Syntax: DSDKPowerSupply*(*getItem)(DSDKPowerSupplyIterator *di);
  Parameters:
  - *di* PowerSupply iterator.

  Returns: The Power Supply at this iterator location.
- **isEnd**
  Description: Returns true if iterator have reached the end.
  Syntax: BOOL(* isEnd)(DSDKPowerSupplyIterator *di);
  Parameters:
  - *di* Power supply iterator.

  Returns: True if have reached the end.

- **next**
  Description: Moves the iterator to the next location.
  Syntax: void(* next)(DSDKPowerSupplyIterator *di);
  Parameters:
  - *di* Power supply iterator.

- **release**
  Description: Releases this object
  Syntax: void(* release)(DSDKPowerSupplyIterator *di);
  Parameters:
  - *di* Power supply iterator.

- **DSDKPowerSupply**
  A structure representing a power supply

  Structure Members
  - hdl - Opaque pointer to power supply specific implementations
  - ft - Pointer to power supply function table.

- **DSDKPowerSupplyFT**
  A structure representing power supply function table.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getTotalPower
- getElementName
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- enablePowerSupply
- disablePowerSupply
- resetPowerSupply
- offlinePowerSupply
- release

Member Function Descriptions

- **getSystemCreationClassName**
  Description: Gets the powersupply System Creation class Name
  Syntax: void (*getSystemCreationClassName) (DSDKPowerSupply* ps, char* name, int max_len);
  Parameters:
  - *ps* Pointer to DSDKPowersupply
  - *name* System Creation Class Name of the powersupply
  - *max_len* maximum buffer length

  Returns none

- **getSystemName**
  Description: Gets SystemName of the powersupply
  Syntax: void (*getSystemName) (DSDKPowerSupply* ps, char* name, int max_len);
  Parameters:

- *ps* Pointer to DSDKPowersupply
- *name* System Name of the powersupply
- *max_len* maximum buffer length

Returns none

- **getCreationClassName**

  Description: Gets the powersupply Creation class Name

  Syntax: void (*getCreationClassName) (DSDKPowerSupply* ps, char* name, int max_len);

  Parameters:
  - *ps* Pointer to DSDKPowersupply
  - *name* Creation class Name of the powersupply
  - *max_len* maximum buffer length

  Returns none

- **getDeviceID**

  Description: Gets DeviceID of the powersupply

  Syntax: void (*getDeviceID) (DSDKPowerSupply* ps, char* devid, int max_len);

  Parameters:
  - *ps* Pointer to DSDKPowersupply
  - *devid* power supply device ID
  - *max_len* maximum buffer length

  Returns none

- **getTotalPower**

  Description: Gets the total power value.

  Syntax: uint32 (*getTotalPower) (DSDKPowerSupply* ps);

  Parameters:
  - *ps* Pointer to DSDKPowersupply

  Returns The total power

- **getElementName**

  Description: Gets ElementName of the powersupply

  Syntax: void (*getElementName) (DSDKPowerSupply* ps, char* name, int max_len);

  Parameters:
  - *ps* Pointer to DSDKPowersupply
  - *name* power supply Element Name

  Returns none

- **getOperationalStatus**

Description: Gets the list of OperationalStatus
Syntax: int (*getOperationalStatus) (DSDKPowersupply* ps, uint16* op_status,int max_types);
Parameters:
- *ps* Pointer to DSDKPowersupply
- *op_status* Operational Status
- *max_types* maximum types

returns: none

- **getOperationalStatusStr**

Description: Gets the list of operational status as string
Syntax: int (*getOperationalStatusStr) (DSDKPowersupply* ps, char** os, int max_os, int max_strlen);
Parameters:
- *ps* Pointer to DSDKPowersupply
- *os* The operational status is filled in.
- *max_os* maximum number of operational status
- *max_strlen* maximum buffer size

Returns: none

- **getHealthState**

Description: Gets the health state of the Powersupply
Syntax: uint16 (*getHealthState) (DSDKPowersupply* ps);
Parameters:
- *ps* Pointer to DSDKPowersupply

Returns: The health state

- **getHealthStateStr**

Description: Gets the health state of the Powersupply as string
Syntax: void (*getHealthStateStr) (DSDKPowersupply* ps, char* state, int max_len);
Parameters:
- *ps* Pointer to DSDKPowersupply
- *state* The health state is filled in.
- *max_len* maximum buffer length

Returns: none

- **getEnabledState**

Description: Gets the state of the Powersupply
Syntax: uint16 (*getEnabledState) (DSDKPowersupply* ps);
Parameters:
- *ps* Pointer to DSDKPowersupply

Returns: The enabled state

- **getEnabledStateStr**

  Description: Gets the state of the Powersupply as string

  Syntax: void (*getEnabledStateStr) (DSDKPowersupply* ps, char* state, int max_len);

  Parameters:

  - *ps* Pointer to DSDKPowersupply
  - *state* The Enabled State is filled in
  - *max_len* maximum buffer length

  Returns: None

- **getRequestedState**

  Description: Gets the requested state of the Powersupply

  Syntax: uint16 (*getRequestedState) (DSDKPowersupply* ps);

  Parameters:

  - ps Pointer to DSDKPowersupply

  Returns: The requested state

- **getRequestedStateStr**

  Description: Gets the requested status of the power supply as string

  Syntax: void (*getRequestedStateStr) (DSDKPowerSupply* ps, char* state, int max_len);

  Parameters:

  - ps Pointer to DSDKPowersupply
  - state Requested state string
  - max_len maxumum buffer length

  Returns None

- **enablePowerSupply**

  Description: Enables this power supply

  Syntax: uint32 (*enablePowerSupply) (DSDKPowerSupply* ps);

  Parameters:

  - *ps* Pointer to DSDKPowersupply

  Returns: 0 on success, throws execption on failure

- **disablePowerSupply**

  Description: Disables this power supply

  Syntax: uint32 (*disablePowerSupply) (DSDKPowerSupply* ps);

  Parameters:

  - *ps* Pointer to DSDKPowersupply

Returns: 0 on success, throws execption on failure

- **resetPowerSupply**

  Description: Resets this power supply
  Syntax: uint32 (*resetPowerSupply) (DSDKPowerSupply* ps);
  Parameters:
  - *ps* Pointer to DSDKPowersupply

  Returns: 0 on success, throws execption on failure

- **offlinePowerSupply**

  Description: Makes power supply offline
  Syntax: uint32 (*offlinePowerSupply) (DSDKPowerSupply* ps);
  Parameters:
  - *ps* Pointer to DSDKPowersupply

  Returns: 0 on success, throws execption on failure

5.2.12. FanRedundancyset

- **DSDKFanRedundancySetIterator**

  A structure representing a fan redundancy set iterator.

  Structure Members

  - hdl - Opaque pointer to fan redundancy set specific implementations
  - ft - Pointer to fan redundancy set iterator function table.

- **enumFanRedundancySets**

  Description: Enumerate all the fan redundancy sets present under a management access point.
  Syntax: DSDKFanRedundancySetIterator* enumFanRedundancySets (DSDKClient* client, BOOL cached);
  Parameters:
  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The fan redundancy set iterator.

- **FanRedundancySetFt**

  A structure representing fan redundancy set iterator function table.

Member Functions

- getItem

- isEnd
- next
- release

Member Function Descriptions

- **getItem**

  Description: Gets the fan redundancy set at this iterator location.

  Syntax: DSDKFanRedundancySet* (*getItem)
  (DSDKFanRedundancySetIterator *di);

  Parameters:

  - *di* Fan redundancy set iterator.

  Returns: The fan redundancy set at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(*isEnd)(DSDKFanRedundancySetIterator *di);

  Parameters:

  - *di* Fan redundancy set iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(*next)(DSDKFanRedundancySetIterator *di);

  Parameters:

  - *di* Fan redundancy set iterator.

- **release**

  Description: Releases this object

  Syntax: void(*release)(DSDKFanRedundancySetIterator *di);

  Parameters:

  - *di* Fan redundancy set iterator.

- **DSDKFanRedundancySet**

  A structure representing a fan redundancy set

  Structure Members

  - hdl - Opaque pointer to fan redundancy set specific implementations
  - ft - Pointer to fan redundancy set function table.

- **DSDKFanRedundancySetFT**

  A structure representing fan redundancy set function table.

Member Functions

- getRedundancyStatus
- getType
- getMinimumNumberNeeded
- failover
- release

Member Function Descriptions

- **getRedundancyStatus**

  Description: Gets the current redundancy status

  Syntax: int (*getRedundancyStatus) (DSDKFanRedundancySet* rs);

  Parameters:

  - *rs* Pointer to Fan redundancy set .

  Returns: The current redundancy status.

- **getType**

  Description: Gets the type of the redundancy set

  Syntax: void (*getType) (DSDKFanRedundancySet* rs, Type_E** type, int max_type);

  Parameters:

  - *rs* Pointer to DSDKFanRedundancySet
  - *type* type of FanRedundancySet

  Returns: The type of the redundancy set.

- **getMinimumNumberNeeded**

  Description: Gets the minimum number needed

  Syntax: uint32 (*getMinimumNumberNeeded) (DSDKFanRedundancySet* rs);

  Parameters:

  - *rs* Pointer to DSDKFanRedundancySet

  Returns: Returns the minimum number needed

- **failover**

  Description: Forces a failover from one fan to another fan.

  Syntax: uint32 (*failover) (DSDKFanRedundancySet* rs, DSDKFan* fan_from, DSDKFan* fan_to);

  Parameters:

  - *rs* Pointer to DSDKFanRedundancySet

  Returns: Returns the status.

- **release**

  Description: Releases this object

  Syntax: void(* release)(DSDKFanRedundancySet *rs);

  Parameters:

  - *rs* Pointer to DSDKFanRedundancySet

5.2.13. PowerSupplyRedundancySet

- **DSDKPowerSupplyRedundancySetIterator**

  A structure representing a power supply redundancy set iterator..

  Structure Members

  - hdl - Opaque pointer to power supply redundancy set specific implementations
  - ft - Pointer to power supply redundancy set iterator function table.

- **enumPowerSupplyRedundancySets**

  Description: Enumerate all the power supply redundancy sets present under a management access point.

  Syntax: DSDKPowerSupplyRedundancySetIterator* enumPowerSupplyRedundancySets (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The power supply redundancy set iterator.

- **PowerSupplyRedundancySetFt**

  A structure representing power supply redundancy set iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Function Descriptions

- **getItem**

  Description: Gets the power supply redundancy set at this iterator

location.

Syntax: DSDKPowerSupplyRedundancySet* (*getItem)
(DSDKPowerSupplyRedundancySetIterator *di);

Parameters:

- *di* Power supply redundancy set iterator.

Returns: The fan redundancy set at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.

  Syntax: BOOL(*isEnd)(DSDKPowerSupplyRedundancySetIterator *di);

  Parameters:

  - *di* Power supply redundancy set iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.

  Syntax: void(*next)(DSDKPowerSupplyRedundancySetIterator *di);

  Parameters:

  - *di* Power supply redundancy set iterator.

- **release**

  Description: Releases this object

  Syntax: void(*release)(DSDKPowerSupplyRedundancySetIterator *di);

  Parameters:

  - *di* Power supply redundancy set iterator.

- **DSDKPowerSupplyRedundancySet**

  A structure representing a fan redundancy set

  Structure Members

  - hdl - Opaque pointer to power supply redundancy set specific implementations
  - ft - Pointer to power supply redundancy set function table.

- **DSDKPowerSupplyRedundancySetFT**

  A structure representing power supply redundancy set function table.

Member Functions

- getInstsnceID
- getRedundancyStatus
- getType

- getMinimumNumberNeeded
- getElementName
- failover
- release

Member Function Descriptions

- **getInstsnceID**

  Description: Gets the InstsnceID of the powersupply

  Syntax: void (*getInstsnceID) (DSDKPowerSupplyRedundancySet* rs, char* ins_id, int max_len);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

  Returns: The InstsnceID of the powersupply

- **getRedundancyStatus**

  Description: Gets the current redundancy status

  Syntax: uint16 (*getRedundancyStatus) (DSDKPowerSupplyRedundancySet* rs);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

  Returns: The current redundancy status

- **getType**

  Description: Gets the type of the redundancy set

  Syntax: void (*getType) (DSDKPowerSupplyRedundancySet* rs, Type_E** type, int max_type);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

  Returns: Returns the type of the redundancy set

- **getMinimumNumberNeeded**

  Description**:** Gets the minimum number needed

  Syntax: uint32 (*getMinimumNumberNeeded) (DSDKPowerSupplyRedundancySet* rs);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

  Returns: Returns the minimum number needed

- **getElementName**

  Description: Gets the ElementName of the redundancy set

  Syntax: void (*getElementName) DSDKPowerSupplyRedundancySet* rs, char* ele_name, int

max_len);

Parameters:

- *rs* Pointer to DSDKPowerSupplyRedundancySet

Returns: The ElementName of the redundancy set

- **failover**

  Description: Forces a failover from one power supply to another power supply.

  Syntax: uint32 (*failover) (DSDKPowerSupplyRedundancySet* rs, DSDKPowerSupply* ps_from, DSDKPowerSupply* ps_to);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

  Returns: The status.

- **release**

  Description: Releases this object

  Syntax: void(* release) (DSDKPowerSupplyRedundancySet *rs);

  Parameters:

  - *rs* Pointer to DSDKPowerSupplyRedundancySet

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

5.2.14. Battery

- **DSDKBatteryIterator**

  A structure representing a battery iterator..

  Structure Members

  - hdl - Opaque pointer to battery specific implementations
  - ft - Pointer to battery iterator function table.

- **enumBattery**

  Description: Enumerate all the batteries present under a management access point.

  Syntax: DSDKBatteryIterator* enumBattery (DSDKClient* client, BOOL cached);

  Parameters:

  - *client* Pointer to the client interface.
  - *cached* Enable/Disable caching.

  Returns: The Battery iterator.

- **BatteryIteratorFt**

  A structure representing Battery iterator function table.

Member Functions

- getItem
- isEnd
- next
- release

Member Function Descriptions

- **getItem**

  Description: Gets the battery at this iterator location.
  Syntax: DSDKBattery*(*getItem)(DSDKBatteryIterator *di);
  Parameters:
  - *di* Battery iterator.

  Returns: The Battery at this iterator location.

- **isEnd**

  Description: Returns true if iterator have reached the end.
  Syntax: BOOL(* isEnd)(DSDKBatteryIterator *di);
  Parameters:
  - *di* Battery iterator.

  Returns: True if have reached the end.

- **next**

  Description: Moves the iterator to the next location.
  Syntax: void(* next)(DSDKBatteryIterator *di);
  Parameters:
  - *di* Battery iterator.

- **release**

  Description: Releases this object
  Syntax: void(* release)(DSDKBatteryIterator *di);
  Parameters:
  - *di* Battery iterator.

- **DSDKBattery**

A structure representing a battery

Structure Members

- hdl - Opaque pointer to battery specific implementations
- ft - Pointer to battery function table.

- **DSDKBatteryFT**

  A structure representing battery function table.

Member Functions

- getSystemCreationClassName
- getSystemName
- getCreationClassName
- getDeviceID
- getBatteryStatus
- getBatterStatusStr
- getOperationalStatus
- getOperationalStatusStr
- getHealthState
- getHealthStateStr
- getEnabledState
- getEnabledStateStr
- getRequestedState
- getRequestedStateStr
- getElementName
- getChemistry
- getMaxRechargeCount
- getRechargeCount
- getExpectedLife
- getEstimatedRunTime
- getTimeToFullCharge
- getMaxRechargeTime
- enable
- disable
- test
- reset
- release

Member Function Descriptions

- **getSystemCreationClassName**

Description: Gets the battery System Creation class Name

Syntax: void (*getSystemCreationClassName) (DSDKBattery* bat, char* name, int max_len);

Parameters:

- *bat* Pointer to Battery
- *name* name of System Creation class
- *max_len* length of the name

Returns: The battery System Creation class Name

- **getSystemName**

Description: Gets the SystemName of the battery

Syntax: void (*getSystemName) (DSDKBattery* bat, char* name, int max_len);

Parameters:

- *bat* Pointer to Battery
- *name* System name
- *max_length* length of the system name

Returns: The SystemName of the battery

- **getCreationClassName**

Description: Gets the battery Creation class Name

Syntax: void (*getCreationClassName) (DSDKBattery* bat, char* name, int max_len);

Parameters:

- *bat* Pointer to Battery
- *name* Creation class Name
- *max_length* length of the Creation class Name

Returns: The battery Creation class Name

- **getDeviceID**

Description: Gets DeviceID of the battery

Syntax: void (*getDeviceID) (DSDKBattery* bat, char* devid, int max_len);

Parameters:

- *bat* Pointer to Battery
- *devid* device id of the battery
- *max_len* length of the devive id name

Returns: The DeviceID of the battery

- **getBatteryStatus**

Description: Gets the status of this battery

Syntax: uint16 (*getBatteryStatus) (DSDKBattery* bat);

Parameters:

- *bat* Pointer to Battery

Returns: The status of battery

- **getBatterStatusStr**

  Description: Gets the status of this battery as string

  Syntax: void (*getBatterStatusStr) (DSDKBattery* bat, char* status, int max_len);

  Parameters:
  - *bat* Pointer to Battery
  - *status* battery status
  - *max_len* length of the status string.

  Returns: The status of this battery as string

- **getOperationalStatus**

  Description: Gets the OperationalStatus

  Syntax: int (*getOperationalStatus) (DSDKBattery* bat, uint16* op_status, int max_types);

  Parameters:
  - *bat* Pointer to Battery
  - *op_status* OperationalStatus

  Returns: The OperationalStatus

- **getOperationalStatusStr**

  Description: Gets the list of operational status

  Syntax: int (*getOperationalStatusStr) (DSDKBattery* bat, char** os, int max_os, int max_strlen);

  Parameters:
  - *bat* Pointer to Battery

  Returns: The list of operational status

- **getHealthState**

  Description: Gets the health state of the batteery

  Syntax: uint16 (*getHealthState) (DSDKBattery* bat);

  Parameters:
  - *bat* Pointer to Battery

  Returns: The health state of the battery

- **getHealthStateStr**

  Description: Gets the health state of this battery

  Syntax: void (*getHealthStateStr) (DSDKBattery* bat, char* state, int max_len);

  Parameters:
  - *bat* Pointer to Battery
  - *state* health state of battery

- *max_len* length of the health state string

Returns: The health state of battery

- **getEnabledState**

Description: Gets the state of this battery

Syntax: uint16 (*getEnabledState) (DSDKBattery* bat);

Parameters:

- *bat* Pointer to Battery

Returns: The state of battery

- **getEnabledStateStr**

Description: Gets the state of the battery as string

Syntax: void (*getEnabledStateStr) (DSDKBattery* bat, char* state, int max_len);

Parameters:

- *bat* Pointer to Battery
- *state* Enabled state of battery
- *max_len* length of the enabled state string

Returns: The state of the battery as string

- **getRequestedState**

Description: Gets the requested state of this battery

Syntax: uint16 (*getRequestedState) (DSDKBattery* bat);

Parameters:

- *bat* Pointer to Battery

Returns: The requested state of battery

- **getRequestedStateStr**

Description: Gets the requested status of this battery as string

Syntax: void (*getRequestedStateStr) (DSDKBattery* bat, char* state, int max_len);

Parameters:

- *bat* Pointer to Battery
- *state* battery state
- *max_len* length of the battery state.

Returns: The requested status of battery

- **getElementName**

Description: Gets the ElemantName of this battery

Syntax: void (*getElementName) (DSDKBattery* bat, char* name, int max_len);

Parameters:

- *bat* Pointer to Battery

Returns: The ElemantName of battery

- **getChemistry**

  Description: Gets the Chemistry of this battery
  Syntax: uint16 (*getChemistry) (DSDKBattery* bat);
  Parameters:
  - *bat* Pointer to Battery

  Returns: The Chemistry of battery

getMaxRechargeCount

Description: Gets MaxRechargeCount of battery
Syntax: uint32 (*getMaxRechargeCount) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The MaxRechargeCount of battery

getRechargeCount

Description: Gets the RechargeCount of this battery
Syntax: uint32 (*getRechargeCount) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The RechargeCount of battery

getExpectedLife

Description: Gets the ExpectedLife of this battery
Syntax: uint32 (*getExpectedLife) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The ExpectedLife of battery

getEstimatedRunTime

Description: Gets the EstimatedRunTime of this battery
Syntax: uint32 (*getEstimatedRunTime) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The EstimatedRunTime of battery

getTimeToFullCharge

Description: Gets the TimeToFullCharge of this battery
Syntax: uint32 (*getTimeToFullCharge) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The TimeToFullCharge of battery

getMaxRechargeTime

Description: Gets the MaxRechargeTime of this battery
Syntax: uint32 (*getMaxRechargeTime) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

Returns: The MaxRechargeTime of battery

enable

Description: Enable/turn on Battery
Syntax: void (*enable) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

disable

Description: Disable/turn off Battery
Syntax: void (*disable) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

test

Description: Test/perform recalculation of charge thresholds.
Syntax: void (*test) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

reset

Description: Reset/recharge of battery.
Syntax: void (*reset) (DSDKBattery* bat);
Parameters:
- *bat* Pointer to Battery

release

Description: Releases this object
Syntax: void(* release)(DSDKBattery *bat);
Parameters:
- *bat* Pointer to DSDKBattery

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- battery_c.h
Library -- dashapic

5.2.15. BIOSManagement

DSDKBIOSAttributeIterator

A structure representing a bios attributes iterator..

Structure Members

hdl - Opaque pointer to bios attributes specific implementations
ft - Pointer to bios attributes iterator function table.

enumBIOSAttributes

Description: Enumerate all the bios attributes present under a management access point.
Syntax: DSDKBIOSAttributeIterator* enumBIOSAttributes (DSDKClient* client, BOOL cached);
Parameters:
- *client* Pointer to the client bios attributes.
- *cached* Enable/Disable caching.

Returns: The bios attributes iterator.

BIOSAttributeIteratorFt

A structure representing bios attributes iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the bios attributes at this iterator location.
Syntax: DSDKBIOSAttribute*(*getItem)(DSDKBIOSAttributeIterator *di);
Parameters:
- *di* bios attributes iterator.

Returns: The biosattributes at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKBIOSAttributeIterator *di);
Parameters:
- *di* bios attributes iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKBIOSAttributeIterator *di);
Parameters:
- *di* bios attributes iterator.

release

Description: Releases this object
Syntax: void(* release)(DSDKBIOSAttributeIterator *di);
Parameters:
- *di* bios attributes iterator.

DSDKBIOSAttribute

A structure representing a biosattributes.

Structure Members

hdl - Opaque pointer to biosattributes specific implementations
ft - Pointer to biosattributes function table.

DSDKBIOSAttributeFT

A structure representing biosattributes function table.

Member Functions

getInstanceID
getAttributeName
getCurrentValue
getDefaultValue
getPendingValue
isReadOnly
isOrderedList
getPossibleValues
getPossibleValuesDescription
getLowerBound
getUpperBound
getProgrammaticUnit

getScalarIncrement
getMaxLength
getMinLength
getStringType
getStringTypeStr
getValueExpression
isPasswordSet
getPasswordEncoding
setAttribute
release

Member Function Descriptions

getInstanceID

Description: Gets the Instance ID of biosattributes

Syntax: void (*getInstanceID) (DSDKBIOSAttribute* ba, char* instanceid, int max_len);

Parameters:

- *ba* Pointer to biosattributes
- *instanceid* instance id of bios
- *max_len* length of the instanceid string

Returns: The Instance ID of biosattributes

getAttributeName

Description: Gets the AttributeName of biosattributes

Syntax: void (*getAttributeName) (DSDKBIOSAttribute* ba, char* name, int max_len);

Parameters:

- *da* Pointer to biosattributes
- *name* attribute name
- *max_len* length of the attribute name

Returns: The AttributeName of biosattributes

getCurrentValue

Description: Gets the Current attribute value of biosattributes

Syntax: int (*getCurrentValue) (DSDKBIOSAttribute* ba, char** current_value,int max_value, int max_strlen);

Parameters:

- *ba* Pointer to biosattributes

- *current_value* current value of bios
- *max_value* maximum value of bios
- *max_strlen* length of the current_value string.

Returns: The Current attribute value of biosattributes

getDefaultValue

Description: Gets the Default attribute value of biosattributes

Syntax: int (*getDefaultValue) (DSDKBIOSAttribute* ba, char** default_values,
int max_value, int max_strlen);

Parameters:

- *ba* Pointer to biosattributes
- *default_values* default value of bios
- *max_value* maximum value of bios
- *max_strlen* length of the default_value string.

Returns: The Default attribute value of biosattributes

getPendingValue

Description: Gets the Pending value of biosattributes

Syntax: int (*getPendingValue) (DSDKBIOSAttribute* ba, char** pending_values, int max_value, int max_strlen);

Parameters:

- *ba* Pointer to biosattributes
- *default_values* default value of bios
- *max_value* maximum value of bios
- *max_strlen* length of the default_value string.

Returns: The Pending value of biosattributes

isReadOnly

Description: Gets IsReadyonly flag of the Attribte

Syntax: BOOL (*isReadOnly) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: true if success /false otherwise

isOrderedList

Description: Gets IsOrderedList

Syntax: BOOL (*isOrderedList) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: True or False

getPossibleValues

Description: Gets The PossibleValues of biosattributes

Syntax: int (*getPossibleValues) (DSDKBIOSAttribute* ba, char** values, int max_value, int max_strlen);

Parameters:

- *ba* Pointer to biosattributes
- *default_values* default value of bios
- *max_value* maximum value of bios
- *max_strlen* length of the default_value string.

Returns: The PossibleValues of biosattributes

getPossibleValuesDescription

Description: Gets the possible description values for this attribute

Syntax: int (*getPossibleValuesDescription) (DSDKBIOSAttribute* ba, char** values, int max_value, int max_strlen);

Parameters:

- *ba* Pointer to biosattributes
- *values* Possible description value of bios
- *max_value* maximum value of bios
- *max_strlen* length of the default_value string.

Returns: The possible description values for this attribute

getLowerBound

Description: Gets lower bound values for this attribute

Syntax: uint64 (*getLowerBound) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The lower bound values for this attribute

getUpperBound

Description: Gets upper bound values for this attribute

Syntax: uint64 (*getUpperBound) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The upper bound values for this attribute

getProgrammaticUnit

Description: Gets the programmatic unit for this attribute

Syntax: void (*getProgrammaticUnit) (DSDKBIOSAttribute* ba, char* program_unit, int max_len);

Parameters:

- *ba* Pointer to biosattributes

- *program_unit* ProgrammaticUnit
- *max_len* length of the program_unit

Returns: The programmatic unit for this attribute

getScalarIncrement

Description: Gets the scalar increment for this attribute

Syntax: uint32 (*getStringType) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The scalar increment for this attribute

getMaxLength

Description: Gets the maximum length of string for this attribute

Syntax: uint32 (*getStringType) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The maximum length of string for this attribute

getMinLength

Description: Gets minimum length of string for this attribute.

Syntax: uint32 (*getStringType) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The minimum length of string for this attribute.

getStringType

Description: Gets the string type of this attribute

Syntax: uint32 (*getStringType) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The string type of this attribute

getStringTypeStr

Description: Gets the string type of attribute as string

Syntax: void (*getStringTypeStr) (DSDKBIOSAttribute* ba, char* type, int max_len);

Parameters:

- *ba* Pointer to biosattributes
- t*ype* string type
- m*ax_len* length of the string type.

Returns: The string type of attribute as string

getValueExpression

Description: Gets the ValueExpression of string for this attribute

Syntax: void (*getValueExpression) (DSDKBIOSAttribute* ba, char* value_exp, int max_len);

Parameters:

- *ba* Pointer to biosattributes

Returns: The ValueExpression of string for this attribute

isPasswordSet

Description: Checks if password is set for this attribute

Syntax: BOOL (*isPasswordSet) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: True or False

getPasswordEncoding

Description: Gets the PasswordEncoding type for this attribute

Syntax: uint32 (*getPasswordEncoding) (DSDKBIOSAttribute* ba);

Parameters:

- *ba* Pointer to biosattributes

Returns: The PasswordEncoding type for this attribute

setAttribute

Description: Sets the biosattributes

Syntax: uint32 (*setAttribute) (DSDKBIOSAttribute* ba, char** values, int num_values);

Parameters:

- *ba* Pointer to biosattributes
- *values* attribute values to set
- *num_values* number of values to be set.

release

Description: Releases this object

Syntax: void(* release)(DSDKBIOSAttribute *ba);

Parameters:

- *ba* Pointer to DSDKBIOSAttributes

DSDKBIOSElementIterator

A structure representing a bios element iterator..

Structure Members

hdl - Opaque pointer to bios element specific implementations
ft - Pointer to bios element iterator function table.

enumBIOSElements

Description: Enumerate all the bios elements present under a management access point.

Syntax: DSDKBIOSElementIterator* enumBIOSElements (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client bios elements.
- *cached* Enable/Disable caching.

Returns: The bios elements iterator.

BIOSElementIteratorFt

A structure representing bios elements iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the bios elements at this iterator location.

Syntax: DSDKBIOSElement*(*getItem)(DSDKBIOSElementIterator *di);

Parameters:

- *di* bios elements iterator.

Returns: The bios elements at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.

Syntax: BOOL(* isEnd)(DSDKBIOSElementIterator *di);

Parameters:

- *di* bios elements iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKBIOSElementIterator *di);

Parameters:

- *di* bios elements iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKBIOSElementIterator *di);

Parameters:

- *di* bios elements iterator.

DSDKBIOSElement

A structure representing a bios elements.

Structure Members

hdl - Opaque pointer to bios elements specific implementations

ft - Pointer to bios elements function table.

DSDKBIOSElementFT

A structure representing bios elements function table.

Member Functions

getManufacturer
getPrimaryBIOS
getVersion
getName
getSoftwareElementState
getSoftwareElementStateStr
getSoftwareElementID
getTargetOperatingSystem
getTargetOperatingSystemStr
getRegistryURIs
getAttributes
restoreDefaults
release

Member Function Descriptions

getManufacturer

Description: Gets the manufacturer of bios

Syntax: void (*getManufacturer) (DSDKBIOSElement* be, char* manufacturer, int max_len);

Parameters:

- *be* Pointer to bios elements
- *manufacturer* name of manufacturer
- *max_len* length of manufacturer name

Returns: Manufacturer of bios

getPrimaryBIOS

Description: Gets the PrimaryBIOS

Syntax: BOOL (*getPrimaryBIOS) (DSDKBIOSElement* be);

Parameters:

- *be* Pointer to bios elements

Returns: True False

getVersion

Description: Gets the version of bios

Syntax: void (*getVersion) (DSDKBIOSElement* be, char* version, int max_len);

Parameters:

- *be* Pointer to bios elements
- *version* version of bios
- *max_len* length of the version name

Returns: The version bios

getName

Description: Gets the name of bios elements

Syntax: void (*getName)(DSDKBIOSElement *be, char * name, int max_len);

Parameters:

- *be* Pointer to bios elements
- *name* name of bios elements
- *max_len* length of the name

Returns: The name of bios elements

getSoftwareElementState

Description: Gets the SoftwareElementState

Syntax: uint16 (*getSoftwareElementState) (DSDKBIOSElement* be);

Parameters:

- *be* Pointer to bios elements

Returns: The SoftwareElementState

getSoftwareElementStateStr

Description: Gets the SoftwareElementState as string

Syntax: void (*getSoftwareElementStateStr) (DSDKBIOSElement* be, char* state, int max_len);

Parameters:

- *be* Pointer to bios elements
- *state* software element state
- *max_len* length of the state string.

Returns: The SoftwareElementState as string

getSoftwareElementID

Description: Gets the SoftwareElementID

Syntax: void (*getSoftwareElementID) (DSDKBIOSElement* be, char* name, int max_len);

Parameters:

- *be* Pointer to bios elements

name software element ID

max_len length of the name string.

Returns: The SoftwareElementID

getTargetOperatingSystem

Description: Gets the TargetOperatingSystem

Syntax: uint16 (*getTargetOperatingSystem) (DSDKBIOSElement* be);

Parameters:

- *be* Pointer to bios elements

Returns: The TargetOperatingSystem

getTargetOperatingSystemStr

Description: Gets the TargetOperatingSystem as string

Syntax: void (*getTargetOperatingSystemStr) (DSDKBIOSElement* be, char* os, int max_len);

Parameters:

- *be* Pointer to bios elements
- os TargetOperatingSystem
- *max_len* length of the os.

Returns: The TargetOperatingSystem as string

getRegistryURIs

Description: Gets the Registry URI's

Syntax: int (*getRegistryURIs) (DSDKBIOSAttribute* ba, char** registryuri, int max_value, int max_strlen);

Parameters:

- *be* Pointer to bios elements
- *registryuri* Registry URI's
- *max_value* maximum value of Registry URI's
- *max_strlen* length of the Registry URI's

Returns: The Registry URI's

getAttributes

Description: Gets the attributes of bios elements

Syntax: int (*getAttributes) (DSDKBIOSElement* be, DSDKBIOSAttribute** attributes, int max_attributes);

Parameters:

- *be* Pointer to bios elements

Returns: The attributes of bios elements

restoreDefaults

Description: restore defaults of the bios elements

Syntax: uint32(*restoreDefaults)(DSDKBIOSElement * be);

Parameters:

- *be* Pointer to bios elements

Returns: Restore defaults of the bios elements

release

Description: Releases this object

Syntax: void(* release)(DSDKBIOSElement *be);

Parameters:

- *be* Pointer to DSDKBIOSEelements

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- biosmanagement_c.h
Library -- dashapic

5.2.16. DHCPClient

DSDKDHCPClientIterator

A structure representing a dhcpclient iterator..

Structure Members

hdl - Opaque pointer to dhcpclient specific implementations
ft - Pointer to dhcpclient iterator function table.

enumDHCPClient

Description: Enumerate all the dhcpclient present under a management access point.

Syntax: DSDKDHCPClientIterator* enumDHCPClient (DSDKClient* client,

BOOL cached);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The dhcpclient iterator.

DHCPClientIteratorFt

A structure representing dhcpclient iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the dhcpclient at this iterator location.
Syntax: DSDKDHCPCLient*(*getItem)(DSDKDHCPCLientIterator *di);
Parameters:

- *di* dhcpclient iterator.

Returns: The dhcpclient at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKDHCPCLientIterator *di);
Parameters:

- *di* dhcpclient iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKDHCPCLientIterator *di);
Parameters:

- *di* dhcpclient iterator.

release

Description: Releases this object
Syntax: void(* release)(DSDKDHCPCLientIterator *di);
Parameters:

- *di* dhcpclient iterator.

DSDKDHCPClient

A structure representing a dhcpclient.

Structure Members

hdl - Opaque pointer to dhcpclient specific implementations
ft - Pointer to dhcpclient function table.

DSDKDHCPCLientFT

A structure representing dhcpclient function table.

Member Functions

getSystemCreationClassName
getSystemName
getCreationClassName
getName
getNameFormat
getProtocolIFType
getProtocolIFTypeStr
getOtherTypeDescription
getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getClientState
getElementName
release

Member Function Descriptions

getSystemCreationClassName

Description: Gets the DHCPClient System Creation class Name
Syntax: void (*getSystemCreationClassName) (DSDKDHCPClient* dhcp, char* name, int max_len);
Parameters:

- *dhcp* Pointer to DHCPClient
- *name* name of System Creation class Name
- *max_len* length of the name.

Returns: The DHCPClient System Creation class Name

getSystemName

Description: Gets the SystemName of the DHCPClient

Syntax: void (*getSystemName) (DSDKDHCPClient* dhcp, char* name, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *name* System Name
- *max_len* length of the name.

Returns: The SystemName of the DHCPClient

getCreationClassName

Description: Gets the DHCPClient Creation class Name

Syntax: void (*getCreationClassName) (DSDKDHCPClient* dhcp, char* name, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *name* Creation class Name
- *max_len* length of the name.

Returns: The DHCPClient Creation class Name

getName

Description: Gets the name of dhcpclient

Syntax: void (*getName) (DSDKDHCPClient* dhcp, char* name, int ` max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *name* name of dhcpclient
- *max_len* length of the name.

Returns: The name of dhcpclient

getNameFormat

Description: Gets The name format

Syntax: void (*getNameFormat) (DSDKDHCPClient* dhcp, char* name_format, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *name_format* name format of dhcpclient
- *max_len* length of the name_format

Returns: The name format

getProtocolIFType

Description: Gets the ProtocolIF type

Syntax: uint16 (*getProtocolIFType) (DSDKDHCPClient* dhcp);

Parameters:

- *dhcp* Pointer to DHCPClient

Returns: The ProtocolIF type

getProtocolIFTypeStr

Description: Gets the ProtocolIF type as string

Syntax: void (*getProtocolIFTypeStr) (DSDKDHCPClient* dhcp, char* type, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *type* type of ProtocolIF
- *max_len* length of the type field.

Returns: The ProtocolIF type as string

getOtherTypeDescription

Description: Gets Protocol IP type description if the ProtocolIPType contains "Other"

Syntax: void (*getOtherTypeDescription) (DSDKDHCPClient* dhcp, char* desc, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *desc* Protocol IP type description
- *max_len* length of the desc field.

Returns: The Protocol IP type description

getEnabledState

Description: Gets the state of dhcpclient

Syntax: uint16 (*getEnabledState) (DSDKDHCPClient* dhcp);

Parameters:

- *dhcp* Pointer to DHCPClient

Returns: The state of dhcpclient

getEnabledStateStr

Description: Gets the state of dhcpclient as string

Syntax: void (*getEnabledStateStr) (DSDKDHCPClient* dhcp, char* state, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *state* state of dhcpclient
- *max_len* length of the state field.

Returns: The state of dhcpclient as string

getRequestedState

Description: Gets the requested state of dhcpclient

Syntax: uint16 (*getRequestedState) (DSDKDHCPClient* dhcp);

Parameters:

- *dhcp* Pointer to DHCPClient

Returns: The requested state of dhcpclient

getRequestedStateStr

Description: Gets the requested state of dhcpclient

Syntax: void (*getRequestedStateStr) (DSDKDHCPClient* dhcp, char* state, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *state* requested state of dhcpclient
- *max_len* length of the state field.

Returns: The requested state of this dhcpclient

getClientState

Description: Gets Client Stateof this dhcpclient

Syntax: void (*getClientState) (DSDKDHCPClient* dhcp, char* client_state, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *client_state* Client Stateof this dhcpclient
- *max_len* length of the client_state

Returns: The Client State of dhcpclient.

getElementName

Description: Gets the Element Name of dhcpclient

Syntax: void (*getElementName) (DSDKDHCPClient* dhcp, char* name, int max_len);

Parameters:

- *dhcp* Pointer to DHCPClient
- *name* Element Name of dhcpclient
- *max_len* length of the element name

Returns: The Element Name of dhcpclient.

release

Description: Releases this object

Syntax: void(* release)(DSDKDHCPClient *dhcp);

Parameters:

- *dhcp* Pointer to DSDKDHCPClient

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- dhcpclient_c.h
Library -- dashapic

5.2.17. DNSClient

DSDKDNSClientIterator

A structure representing a dnsclient iterator..

Structure Members

hdl - Opaque pointer to dnsclient specific implementations
ft - Pointer to dnsclient iterator function table.

enumDNSClient

Description: Enumerate all the dnsclient present under a management access point.

Syntax: DSDKDNSClientIterator* enumDNSClient (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The dnsclient iterator.

DNSClientIteratorFt

A structure representing dnsclient iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the dnsclient at this iterator location.

Syntax: DSDKDNSCLient*(*getItem)(DSDKDNSCLientIterator *di);

Parameters:

- *di* dnsclient iterator.

Returns: The dnsclient at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.

Syntax: BOOL(* isEnd)(DSDKDNSCLientIterator *di);

Parameters:

- *di* dnsclient iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKDNSCLientIterator *di);

Parameters:

- *di* dnsclient iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKDNSCLientIterator *di);

Parameters:

- *di* dnsclient iterator.

DSDKDNSClient

A structure representing a dnsclient.

Structure Members

hdl - Opaque pointer to dnsclient specific implementations

ft - Pointer to dnsclient function table.

DSDKDNSCLientFT

A structure representing dnsclient function table.

Member Functions

getSystemCreationClassName
getSystemName
getCreationClassName
getName
getNameFormat
getHostname
getProtocolIFType
getProtocolIFTypeStr

getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getElementName
appendPrimarySuffixes
appendParentSuffixes
getDNSSuffixesToAppend
getDomainName
useSuffixWhenRegistering
registerThisConnectionsAddress
getDHCPOptionsToUse
release

Member Function Descriptions

getSystemCreationClassName

Description: Gets the SystemCreationClassName of dnsclient
Syntax: void (*getSystemCreationClassName) (DSDKDNSClient* dns, char* name, int max_len);
Parameters:
- *dns* Pointer to DNSClient

Returns: The SystemCreationClassName of dnsclient

getSystemName

Description: Gets the SystemName dnsclient
Syntax: void (*getSystemName) (DSDKDNSClient* dns, char* name, int max_len);
Parameters:
- *dns* Pointer to DNSClient

Returns: The SystemName of this dnsclient

getCreationClassName

Description: Gets the CreationClassName dnsclient
Syntax: void (*getCreationClassName) (DSDKDNSClient* dns, char* name, int max_len);
Parameters:
- *dns* Pointer to DNSClient

Returns: The CreationClassName of this dnsclient

getName

Description: Gets the Name dnsclient
Syntax: void (*getName) (DSDKDNSClient* dns, char* name, int max_len);
Parameters:

- *dns* Pointer to DNSClient

Returns: The Name of this dnsclient

getNameFormat

Description: Gets the NameFormat

Syntax: void (*getNameFormat) (DSDKDNSClient* dns, char* name_format, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The NameFormat

getHostname

Description: Gets the Hostname

Syntax: void (*getHostname) (DSDKDNSClient* dns, char* hostname, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The Hostname

getProtocolIFType

Description: Gets the ProtocolIFType

Syntax: uint16 (*getProtocolIFType) (DSDKDNSClient* dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: The ProtocolIFType

getProtocolIFTypeStr

Description: Gets the ProtocolIFType as string

Syntax: void (*getProtocolIFTypeStr) (DSDKDNSClient* dns, char* type, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The ProtocolIFType

getEnabledState

Description: Gets the enabled state of dnsclient

Syntax: uint16 (*getEnabledState)(DSDKDNSClient *dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: The enabled state.

getEnabledStateStr

Description: Gets the enabled state as string

Syntax: void (*getEnabledStateStr) (DSDKDNSClient* dns, char* state, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The enabled state.

getRequestedState

Description: Gets the requestedState of dnsclient

Syntax: uint16 (*getRequestedState) (DSDKDNSClient* dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: The requestedState.

getRequestedStateStr

Description: Gets the requestedState as string

Syntax: void (*getRequestedStateStr) (DSDKDNSClient* dns, char* state, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The requestedState.

getAppendPrimarySuffixes

Description: Gets the AppendPrimarySuffixes

Syntax: BOOL (*getAppendPrimarySuffixes) (DSDKDNSClient *dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: true if success
false otherwise

getAppendParentSuffixes

Description: Gets the AppendParentSuffixes

Syntax: BOOL (*getAppendParentSuffixes) (DSDKDNSClient *dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: true if success
false otherwise

getDNSSuffixesToAppend

Description: Gets the value DNSSuffixesToAppend

Syntax: int (*getDNSSuffixesToAppend) (DSDKDNSClient* dns, char** append_value, int max_value, int max_strlen);

Parameters:

- *dns* Pointer to DNSClient

Returns: The value DNSSuffixesToAppend

getDomainName

Description: Gets the domain name of dnsclient

Syntax: void (*getDomainName)(DSDKDNSClient *dns, char * name, int max_len);

Parameters:

- *dns* Pointer to DNSClient

Returns: The domain name of dnsclient.

useSuffixWhenRegistering

Description: Gets the useSuffixWhenRegistering

Syntax: BOOL (*useSuffixWhenRegistering) (DSDKDNSClient* dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: true if success
false otherwise

registerThisConnectionsAddress

Description: Gets the registerThisConnectionsAddress

Syntax: BOOL (*registerThisConnectionsAddress) (DSDKDNSClient *dns);

Parameters:

- *dns* Pointer to DNSClient

Returns: true if success
false otherwise

getDHCPOptionsToUse

Description: Gets the list of DHCPOptionsToUse

Syntax: int (*getDHCPOptionsToUse) (DSDKDNSClient* dns, uint16* options, int max_types);

Parameters:

- *dns* Pointer to DNSClient

Returns: The list of DHCPOptionsToUse

release

Description: Releases this object

Syntax: void(* release)(DSDKDNSClient *dns);

Parameters:

- *dns* Pointer to DSDKDNSClient

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to

get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- dnsclient_c.h
Library -- dashapic

5.2.18. IPInterface

DSDKIPInterfaceIterator

A structure representing a ipinterface iterator..

Structure Members

hdl - Opaque pointer to ipinterface specific implementations
ft - Pointer to ipinterface iterator function table.

enumIPInterface

Description: Enumerate all the ipinterface present under a management access point.

Syntax: DSDKIPInterfaceIterator* enumIPInterface (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The ipinterface iterator.

IPInterfaceIteratorFt

A structure representing ipinterface iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the ipinterface at this iterator location.

Syntax: DSDKIPInterface*(*getItem)(DSDKIPInterfaceIterator *di);

Parameters:

- *di* ipinterface iterator.

Returns: The ipinterface at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.

Syntax: BOOL(* isEnd)(DSDKIPInterfaceIterator *di);

Parameters:

- *di* ipinterface iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKIPInterfaceIterator *di);

Parameters:

- *di* ipinterface iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKIPInterfaceIterator *di);

Parameters:

- *di* ipinterface iterator.

DSDKIPInterface

A structure representing a ipinterface.

Structure Members

hdl - Opaque pointer to ipinterface specific implementations
ft - Pointer to ipinterface function table.

DSDKIPInterfaceFT

A structure representing ipinterface function table.

Member Functions

getSystemCreationClassName
getSystemName
getCreationClassName
getName
getNameFormat
getProtocolIFType
getProtocolIFTypeStr

getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getElementName
getIPv4Address
getSubnetMask
getAddressOrigin
getAddressOriginStr
getIPv6Address
getIPv6AddressType
getIPv6AddressTypeStr
getIPv6SubnetPrefixLength
release

Member Function Descriptions

getSystemCreationClassName

Description: Gets the SystemCreationClassName of ipinterface

Syntax: void (*getSystemCreationClassName) (DSDKIPInterface *IP* dns, char* name, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The SystemCreationClassName of IPInterface

getSystemName

Description: Gets the SystemName ipinterface

Syntax: void (*getSystemName) (DSDKIPInterface *IP, char* name, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The SystemName of this ipinterface

getCreationClassName

Description: Gets the CreationClassName ipinterface

Syntax: void (*getCreationClassName) (DSDKIPInterface* IP, char* name, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The CreationClassName of this ipinterface

getName

Description: Gets the Name ipinterface

Syntax: void (*getName) (DSDKIPInterface* IP, char* name, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Name of this ipinterface

getNameFormat

Description: Gets the NameFormat

Syntax: void (*getNameFormat) (DSDKIPInterface* IP, char* name_format, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The NameFormat

getHostname

Description: Gets the Hostname

Syntax: void (*getHostname) (DSDKIPInterface* IP, char* hostname, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Hostname

getProtocolIFType

Description: Gets the ProtocolIFType

Syntax: uint16 (*getProtocolIFType) (DSDKIPInterface* IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: The ProtocolIFType

getProtocolIFTypeStr

Description: Gets the ProtocolIFType as string

Syntax: void (*getProtocolIFTypeStr) (DSDKIPInterface* IP, char* type, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The ProtocolIFType

getEnabledState

Description: Gets the enabled state of ipinterface

Syntax: uint16 (*getEnabledState)(DSDKIPInterface *IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: The enabled state.

getEnabledStateStr

Description: Gets the enabled state as string

Syntax: void (*getEnabledStateStr) (DSDKIPInterface* IP, char* state, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The enabled state.

getRequestedState

Description: Gets the requestedState of ipinterface

Syntax: uint16 (*getRequestedState) (DSDKIPInterface* IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: The requestedState.

getRequestedStateStr

Description: Gets the requestedState as string

Syntax: void (*getRequestedStateStr) (DSDKIPInterface* IP, char* state, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The requestedState.

getElementName

Description: Gets the ElementName

Syntax: void (*getElementName) (DSDKIPInterface* Ip,char* ele_name,int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The ElementName of this ipinterface

getIPv4Address

Description: Gets the IPv4Address

Syntax: void (*getIPv4Address) (DSDKIPInterface* IP, char* addr, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The IPv4Address

getSubnetMask

Description: Gets the getSubnetMask

Syntax: void (*getSubnetMask) (DSDKIPInterface* IP, char* sub_mask, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The subnetmask

getAddressOrigin

Description: Gets the AddressOrigin

Syntax: uint16 (*getAddressOrigin) (DSDKIPInterface* IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: The AddressOrigin

getAddressOriginStr

Description: Gets the AddressOrigin as string

Syntax: void (*getAddressOriginStr) (DSDKIPInterface* IP, char* orgin, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The AddressOrigin

getIPv6Address

Description: Gets the IPv6Address

Syntax: void (*getIPv6Address) (DSDKIPInterface* IP, char* addr, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Ipv6Address

getIPv6Address

Description: Gets the IPv6Address

Syntax: void (*getIPv6Address) (DSDKIPInterface* IP, char* addr, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Ipv6Address

getIPv6AddressType

Description: Gets the IPv6AddressType

Syntax: uint16 (*getIPv6AddressType) (DSDKIPInterface* IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: Ipv6AddressType

getIPv6AddressTypeStr

Description: Gets the Ipv6AddressType as string

Syntax: void (*getIPv6AddressTypeStr) (DSDKIPInterface* IP, char* type, int max_len);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Ipv6AddressType

getIPv6SubnetPrefixLength

Description: Gets the IPv6SubnetPrefixLength

Syntax: uint16 (*getIPv6SubnetPrefixLength) (DSDKIPInterface* IP);

Parameters:

- *IP* Pointer to IPInterface

Returns: The Ipv6SubnetPrefixLength

release

Description: Releases this object

Syntax: void(* release)(DSDKIPInterface *IP);

Parameters:

- *IP* Pointer to DSDKIPInterface

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- ipinterface_c.h
Library -- dashapic

5.2.19. NetworkPort

DSDKNetworkPortIterator

A structure representing a networkport iterator.

Structure Members

hdl - Opaque pointer to networkport specific implementations
ft - Pointer to networkport iterator function table.

enumNetworkPorts

Description: Enumerate all the networkport present under a management access point.

Syntax: DSDKNetworkPortIterator* enumNetworkPorts (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client networkport.
- *cached* Enable/Disable caching.

Returns: The networkport iterator.

NetworkPortIteratorFt

A structure representing networkport iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the networkport at this iterator location.
Syntax: DSDKNetworkPort*(*getItem)(DSDKNetworkPortIterator *di);
Parameters:

- *di* networkport iterator.

Returns: The networkport at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKNetworkPortIterator *di);
Parameters:

- *di* networkport iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKNetworkPortIterator *di);
Parameters:

- *di* networkport iterator.

release

Description: Releases this object
Syntax: void(* release)(DSDKNetworkPortIterator *di);
Parameters:

- *di* networkport iterator.

DSDKNetworkPort

A structure representing a networkport.

Structure Members

hdl - Opaque pointer to networkport specific implementations
ft - Pointer to networkport function table.

DSDKNetworkPortFT

A structure representing networkport function table.

Member Functions

getSystemCreationClassName
getSystemName
getCreationClassName
getName
getSpeed
getLinkTechnology
getLinkTechnologyStr
getPermanentAddress
getMaxSpeed
getRequestedSpeed
getDeviceID
getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getElementName
release

Member Function Descriptions

getSystemCreationClassName

Description: Gets the SystemCreationClassName of networkport
Syntax: void (*getSystemCreationClassName) (DSDKNetworkPort *np, char*name, int max_len);
Parameters:
- *np* Pointer to NetworkPort

Returns: The SystemCreationClassName of networkport

getCreationClassName

Description: Gets the CreationClassName NetworkPort

Syntax: void (*getCreationClassName) (DSDKNetworkPort *np, char* name, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The CreationClassName of this NetworkPort

getSystemName

Description: Gets the SystemName networkport

Syntax: void (*getSystemName) (DSDKNetworkPort *np, char* name, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The SystemName of this NetworkPort

getName

Description: Gets the Name networkport

Syntax: void (*getName) (DSDKNetworkPort *np, char* name, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The Name of this networkport

getSpeed

Description: Gets the speed

Syntax: uint64 (*getSpeed) (DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The speed

getLinkTechnology

Description: Gets the LinkTechnology

Syntax: uint16 (*getLinkTechnology) (DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The LinkTechnology

getLinkTechnologyStr

Description: Gets the LinkTechnology as string

Syntax: void (*getLinkTechnologyStr) (DSDKNetworkPort* np, char* lt, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The LinkTechnology

getPermanentAddress

Description: Gets the PermanentAddress

Syntax: void (*getPermanentAddress) (DSDKNetworkPort* np, char* perm_addr, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The PermanentAddress

getMaxSpeed

Description: Gets MaxSpeed

Syntax: uint64 (*getMaxSpeed) (DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The MaxSpeed

getRequestedSpeed

Description: Gets RequestedSpeed

Syntax: uint64 (*getRequestedSpeed) (DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The RequestedSpeed

getDeviceID

Description: Gets DeviceID

Syntax: void (*getDeviceID) (DSDKNetworkPort* np, char* devid, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The DeviceID

getEnabledState

Description: Gets the enabled state of networkport

Syntax: uint16 (*getEnabledState)(DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The EnabledState

getEnabledStatestr

Description: Gets the enabled state of networkport as string

Syntax: void (*getEnabledStateStr) (DSDKNetworkPort* np, char* state, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The EnabledState

getRequestedState

Description: Gets the requestedState of networkport

Syntax: uint16 (*getRequestedState) (DSDKNetworkPort* np);

Parameters:

- *np* Pointer to NetworkPort

Returns: The requestedState.

getRequestedStateStr

Description: Gets the requestedState as string

Syntax: void (*getRequestedStateStr) (DSDKNetworkPort* np, char* state, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The requestedState.

getElementName

Description: Gets ElementName

Syntax: void (*getElementName) (DSDKNetworkPort* np, char* ele_name, int max_len);

Parameters:

- *np* Pointer to NetworkPort

Returns: The ElementName

release

Description: Releases this object

Syntax: void(* release)(DSDKNetworkPort *np);

Parameters:

- *np* Pointer to DSDKNetworkPort

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- networkport_c.h

Library -- dashapic

5.2.20. OpaqueManagementData

DSDKOpaqueManagementDataIterator

A structure representing a opaquemanagementdata iterator.

Structure Members

hdl - Opaque pointer to opaquemanagementdata specific implementations
ft - Pointer to opaquemanagementdata iterator function table.

enumOaqueManagementData

Description: Enumerate all the opaquemanagementdata present under a management access point.

Syntax:
```
DSDKOpaqueManagementDataIterator* enumOpaqueManagementData (
                                        DSDKClient* client,
                                    BOOL cached);
```

Parameters:

- *client* Pointer to the client opaquemanagementdata.
- *cached* Enable/Disable caching.

Returns: The opaquemanagementdata iterator.

OpaqueManagementDataIteratorFt

A structure representing opaquemanagementdata iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the opaquemanagementdata at this iterator location.

Syntax:
```
DSDKOpaqueManagementData*(*getItem)
                ( DSDKOpaqueManagementDataIterator *di);
```

Parameters:

- *di* opaquemanagementdata iterator.

Returns: The opaquemanagementdata at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.

Syntax:
```
BOOL(* isEnd)( DSDKOpaqueManagementDataIterator *di);
```

Parameters:

- *di* opaquemanagementdata iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKOpaqueManagementDataIterator *di);

Parameters:

- *di* opaquemanagementdata iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKOpaqueManagementDataIterator *di);

Parameters:

- *di* opaquemanagementdata iterator.

DSDKOpaqueManagementData

A structure representing a opaquemanagementdata.

Structure Members

hdl - Opaque pointer to networkport specific implementations
ft - Pointer to opaquemanagementdata function table.

DSDKOpaqueManagementDataFT

A structure representing opaquemanagementdata function table.

Member Functions

getTransformedDataSize
getMaxSize
getUntransformedDataFormat
getTransformations
getTransformationKeyIDs
getWriteLimited
getDataOrganization
getAccess
getNumberOfBlocks
getBlockSize
getConsumableBlocks
getSystemCreationClassName
getSystemName

getCreationClassName
getDeviceID
readData
writeData
release

Member Function Descriptions

getTransformedDataSize

Description: Gets the transformed data size of opaquemanagementdata
Syntax: uint64 (*getTransformedDataSize)(DSDKOpaqueManagementData *omd);
Parameters:
- *omd* Pointer to opaquemanagementdata

Returns: The TransformedDataSize

getMaxSize

Description: Gets the maximum size of opaquemanagementdata
Syntax: uint64 (*getMaxSize)(DSDKOpaqueManagementData *omd);
Parameters:
- *omd* Pointer to opaquemanagementdata

Returns: The MaxSize

getUntransformedDataFormat

Description: Gets the data format of opaquemanagementdata
Syntax: void (*getUntransformedDataFormat) (DSDKOpaqueManagementData *omd, char *str, int max_len);
Parameters:
- *omd* Pointer to opaquemanagementdata

Returns: The UntransformedDataFormat

getTransformations

Description: Gets array of Transformations
Syntax: int (*getTransformations) (DSDKOpaqueManagementData* omd, uint16* transformations, int max_transform);
Parameters:
- *omd* Pointer to opaquemanagementdata

Returns: The list of Transformations

getTransformationKeyIDs

Description: Gets an array of TransformationKeyIDs
Syntax: int (*getTransformationKeyIDs) (DSDKOpaqueManagementData* omd, char** transIDs, int max_ids, int max_strlen);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The array of TransformationKeyIDs

getWriteLimited

Description: Gets the write limit of opaquemanagementdata

Syntax: BOOL (*getWriteLimited)(DSDKOpaqueManagementData *omd);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The write limit of this opaquemanagementdata

getDataOrganization

Description: Gets the DataOrganization

Syntax: uint16 (*getDataOrganization) (DSDKOpaqueManagementData* omd);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The DataOrganization

getAccess

Description: Gets the Access data

Syntax: uint16 (*getAccess) (DSDKOpaqueManagementData* omd);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The Access data

getNumberOfBlocks

Description: Gets the number of blocks in opaquemanagementdata

Syntax: uint64 (*getNumberOfBlocks)(DSDKOpaqueManagementData **omd);**

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The NumberOfBlocks

getBlockSize

Description: Gets the BlockSize

Syntax: uint64 (*getBlockSize) (DSDKOpaqueManagementData* omd);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The BlockSize

getConsumableBlocks

Description: Gets the ConsumableBlocks

Syntax: uint64 (*getConsumableBlocks) (DSDKOpaqueManagementData* omd);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The ConsumableBlocks

getSystemCreationClassName

Description: Gets the System Creation class of the opaquemangementdata

Syntax: string getSystemCreationClassName (void);

Returns: The System Creation Class name

getSystemName

Description: Gets the System name of the opaquemanagementdata

Syntax: string getSystemName (void);

Returns: The System name

getCreationClassName

Description: Gets the CreationClassName

Syntax: string getCreationClassName (void) const;

Returns: The Creation class name

getDeviceID

Description: Gets the device id of opaquemanagementdata

Syntax: void (*getDevicdID)(DSDKOpaqueManagementData *omd, char *str, int max_len);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The deviceid of opaquemanagementdata

readData

Description: read data of opaquemanagementdata

Syntax: uint32(*readData)(DSDKOpaqueManagementData *omd, uint64 offset , uint64 length);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The read status of opaquemanagementdata

writeData

Description: write data of opaquemanagementdata

Syntax: uint32(*writeData)(DSDKOpaqueManagementData *omd,

uint64 offset,uint64 length,uint8 *data);

Parameters:

- *omd* Pointer to opaquemanagementdata

Returns: The write status of opaquemanagementdata

release

Description: Releases this object

Syntax: void(* release)(DSDKOpaqueManagementData *omd);

Parameters:

- *omd* Pointer to DSDKOpaqueManagementData

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- opaquemanagementdata_c.h
Library -- dashapic

5.2.21. OperatingSystem

DSDKOperatingSystemIterator

A structure representing a operatingsystem iterator.

Structure Members

hdl - Opaque pointer to operatingsystem specific implementations
ft - Pointer to operatingsystem iterator function table.

enumOperatingSystem

Description: Enumerate all the operatingsystem present under a management access point.

Syntax: DSDKOperatingSystemIterator* enumOperatingSystem (DSDKClient* client,BOOL cached);

Parameters:

- *client* Pointer to the client operatingsystem.
- *cached* Enable/Disable caching.

Returns: The operatingsystem iterator.

OperatingSystemIteratorFt

A structure representing operatingsystem iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the operatingsystem at this iterator location.
Syntax: DSDKOperatingSystem*(*getItem)
(DSDKOperatingSystemIterator *di);
Parameters:
- *di* operatingsystem iterator.

Returns: The operatingsystem at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKOperatingSystemIterator *di);
Parameters:
- *di* operatingsystem iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKOperatingSystemIterator *di);
Parameters:
- *di* operatingsystem iterator.

release

Description: Releases this object
Syntax: void(* release)(DSDKOperatingSystemIterator *di);
Parameters:
- *di* operatingsystem iterator.

DSDKOperatingSystem

A structure representing a operatingsystem.

Structure Members

hdl - Opaque pointer to operatingsystem specific implementations
ft - Pointer to operatingsystem function table.

DSDKOperatingSystemFT

A structure representing operatingsystem function table.

Member Functions

getCSCreationClassName
getCSName
getCreationClassName
getName
getOStype
getOtherTypeDescription
getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getAvailableRequestedStates
getAvailableRequestedStatesStr
getTransitioningToState
getTransitioningToStateStr
release

Member Function Descriptions

getCSCreationClassName

Description: Gets the CSCreationClassName of operatingsystem
Syntax: void (*getCSCreationClassName) (DSDKOperatingSystem* os, char* name, int max_len);
Parameters:
- *os* Pointer to operatingsystem

Returns: The CSCreationClassName of operatingsystem

getCSName

Description: Gets the CSName of operatingsystem
Syntax: void (*getCSName) (DSDKOperatingSystem* os, char* name,int max_len);
Parameters:
- *os* Pointer to operatingsystem

Returns: The CSName of operatingsystem

getCreationClassName

Description: Gets the CreationClassName of operatingsystem

Syntax: void (*getCreationClassName) (DSDKOperatingSystem* os, char* name,int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The CreationClassName of operatingsystem

getName

Description: Gets the name of operatingsystem

Syntax: void (*getName)(DSDKOperatingSystem *os, char * name, int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The name of operatingsystem

getOStype

Description: Gets the Os type of operatingsystem

Syntax: void (*getOStype)(DSDKOperatingSystem *os, char * type, int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The os type of operatingsystem

getOtherTypeDescription

Description: Gets the OtherTypeDescription of operatingsystem

Syntax: void (*getOtherTypeDescription) (DSDKOperatingSystem* os, char* type, int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The OtherTypeDescription of operatingsystem

getEnabledState

Description: Gets the enabled state of operatingsystem

Syntax: uint16 (*getEnabledState)(DSDKOperatingSystem* os);

Parameters:

- *os* Pointer to operatingsystem

Returns: The EnabledState.

getEnabledStateStr

Description: Gets the enabled state of operatingsystem as string

Syntax: void (*getEnabledStateStr) (DSDKOperatingSystem* os, char* state, int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The EnabledState.

getRequestedState

Description: Gets the requestedState of operatingsystme

Syntax: uint16 (*getRequestedState) (DSDKOperatingSystem* os);

Parameters:

- *os* Pointer to operatingsystem

Returns: The requestedState.

getRequestedStateStr

Description: Gets the requestedState as string

Syntax: void (*getRequestedStateStr) (DSDKOperatingSystem* os, char* state,int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The requestedState.

getAvailableRequestedStates

Description: Gets the AvailableRequestedStates of operatingsystem

Syntax: int (*getAvailableRequestedStatesStr) (DSDKOperatingSystem* os, char** states, int max_states, int max_strlen);

Parameters:

- *os* Pointer to operatingsystem

Returns: The available resquested state

getTransitioningToState

Description: Gets the TransitioningToState of operatingsystem

Syntax: uint16 (*getTransitioningToState) (DSDKOperatingSystem* os);

Parameters:

- *os* Pointer to operatingsystem

Returns: The transitioning state of operating system

getTransitioningToStateStr

Description: Gets the TransitioningToStateStr of operatingsystem as string

Syntax: void (*getTransitioningToStateStr) (DSDKOperatingSystem* os, char* state, int max_len);

Parameters:

- *os* Pointer to operatingsystem

Returns: The TransitioningToState

release

Description: Releases this object

Syntax: void(* release)(DSDKOperatingSystem *os);

Parameters:

- *os* Pointer to operatingsystem

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- operatingsystem_c.h
Library -- dashapic

5.2.22. TextRedirection

DSDKTextRedirectionIterator

A structure representing a textredirection iterator.

Structure Members

hdl - Opaque pointer to textredirection specific implementations
ft - Pointer to textredirection iterator function table

enumTextRedirection

Description: Enumerate all the textredirection present under a management access point.

Syntax: DSDKTextRedirectionIterator* enumTextRedirection (DSDKClient* client, BOOL cached);

Parameters:

- *client* Pointer to the client textredirection
- *cached* Enable/Disable caching.

Returns: The textredirection iterator.

TextRedirectionIteratorFt

A structure representing textredirection iterator function table.

Member Functions

getItem
isEnd
next

release

Member Functions Description

getItem

Description: Gets the textredirection at this iterator location.
Syntax: DSDKTextRedirection*(*getItem)(DSDKTextRedirectionIterator *di);
Parameters:
- *di* textredirection iterator.

Returns: The textredirection at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKTextRedirectionIterator *di);
Parameters:
- *di* textredirection iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKTextRedirectionIterator *di);
Parameters:
- *di* textredirection iterator.

release

Description: Releases this object
Syntax: void(* release)(DSDKTextRedirectionIterator *di);
Parameters:
- *di* textredirection iterator.

DSDKTextRedirection

A structure representing a textredirection.

Structure Members

hdl - Opaque pointer to textredirection specific implementations
ft - Pointer to textredirection function table.

DSDKTextRedirectionFT

A structure representing textredirection function table.

Member Functions

getSystemCreationClassName
getSystemName
getName
getCreationClassName
getElementName
getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getTerminationSequence
getTextFlowType
getPortNumber
getProtocolIFType
activate
enable
disable
startRedirection
release

Member Functions Description

getSystemCreationClassName

Description: Gets SystemCreationClassName

Syntax: void (*getSystemCreationClassName) (DSDKTextRedirection* tr, char*str, int max_len);

Returns: The system creation classname

getSystemName

Description: Gets SystemName

Syntax: void (*getSystemName) (DSDKTextRedirection* tr, char* str, int max_len);

Returns: The system name

getName

Description: Gets the Text redirection name

Syntax: void (*getName) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.

- *str* The name stringm
- *max_len* maximum buffer size

getCreationClassName

Description: Gets CreationClassName

Syntax: void (*getCreationClassName) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *str* The name stringm
- *max_len* maximum buffer size

Returns: the creationclassname

getElementName

Description: Gets ElementName

Syntax: void (*getElementName) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *str* The name stringm
- *max_len* maximum buffer size

Returns: The ElementName

getEnabledState

Description: Gets the state of the text redirection

Syntax: uint16 (*getEnabledState) (DSDKTextRedirection* tr);

Returns: The enabled state

getEnabledStateStr

Description: Gets the state of the text redirection as string

Syntax: void (*getEnabledStateStr) (DSDKTextRedirection* tr, char* state, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *state* The Enabled State is filled in
- *max_len* maximum buffer length

getRequestedState

Description: Gets the requested state of the text redirection

Syntax: uint16 (*getRequestedState) (DSDKTextRedirection* tr);

Returns: The enabled state

getRequestedStateStr

Description: Gets the requested status of the text redirection as string

Syntax: void (*getRequestedStateStr) (DSDKTextRedirection* tr, char* state, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *state* The Enabled State is filled in
- *max_len* maximum buffer length

getTerminationSequence

Description: Gets the session terminate sequence

Syntax: void (*getTerminationSequence) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *str* The termination sequence string
- *max_len* maximum buffer size

getTextFlowType

Description: Gets the text flow type

Syntax: void (*getTextFlowType) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *str* The textflowtype string
- *max_len* maximum buffer size

getPortNumber

Description: Gets the port number

Syntax: uint32 (*getPortNumber) (DSDKTextRedirection* tr);

Returns: The port number

getProtocolIFType

Description: Gets protocol interface type

Syntax: void (*getProtocolIFType) (DSDKTextRedirection* tr, char* str, int max_len);

Parameters:

- *tr* pointer to textredirection.
- *str* The protocol interface type
- *max_len* maximim buffer length

activate

Description: Enable/Activate this redirection session

Syntax: void (*activate) (DSDKTextRedirection* tr);

Parameters:

- *tr* pointer to textredirection.

enable

Description: Enable this redirection session.
Syntax: void (*enable) (DSDKTextRedirection* tr);
Parameters:

- *tr* pointer to textredirection.

disable

Description: Disable this redirection session
Syntax: void (*disable) (DSDKTextRedirection* tr);
Parameters:

- *tr* pointer to textredirection.

startRedirection

Description: Starts the text redirection
Syntax: void (*startRedirection) (DSDKTextRedirection* tr);
Parameters:

- *tr* pointer to textredirection.

release

Description: Releases this object
Syntax: void(* release)(DSDKTextRedirectionIterator *di);
Parameters:

- *tr* pointer to textredirection.

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- textredirection_c.h
Library -- dashapic

5.2.23. USBRedirection

DSDKUSBRedirectionIterator

A structure representing a usbredirection iterator.

Structure Members

hdl - Opaque pointer to usbredirection specific implementations
ft - Pointer to usbredirection iterator function table.

enumUSBRedirections

Description: Enumerate all the usbredirection present under a management access point.
Syntax: DSDKUSBRedirectionIterator* enumUSBRedirections (DSDKClient* client, BOOL cached);
Parameters:
- *client* Pointer to the client usbredirection
- *cached* Enable/Disable caching.

Returns: The usbredirection iterator.

USBRedirectionIteratorFt

A structure representing usbredirection iterator function table.

Member Functions

getItem
isEnd
next
release

Member Functions Description

getItem

Description: Gets the usbredirection at this iterator location.
Syntax: DSDKUSBRedirection*(*getItem)(DSDKUSBRedirectionIterator *di);
Parameters:
- *di* usbredirection iterator.

Returns: The usbredirection at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKUSBRedirectionIterator *di);
Parameters:
- *di* usbredirection iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.
Syntax: void(* next)(DSDKUSBRedirectionIterator *di);
Parameters:

- *di* usbredirection iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKUSBRedirectionIterator *di);

Parameters:

- *di* usbredirection iterator.

DSDKUSBRedirection

A structure representing a usbredirection.

Structure Members

hdl - Opaque pointer to usbredirection specific implementations

ft - Pointer to usbredirection function table.

DSDKUSBRedirectionFT

A structure representing usbredirection function table.

Member Functions

getSystemCreationClassName
getSystemName
getName
getCreationClassName
getElementName
getEnabledState
getEnabledStateStr
getRequestedState
getRequestedStateStr
getConnectionMode
activate
enable
disable
release

Member Functions Description

getSystemCreationClassName

Description: Gets SystemCreationClassName

Syntax: void (*getSystemCreationClassName) (DSDKUSBRedirection* usbr, char* str, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *str* The name string
- *max_len* maximum buffer size

Returns: The System creation classname

getSystemName

Description: Gets SystemName

Syntax: void (*getSystemName) (DSDKUSBRedirection* usbr, char* str, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *str* The name string
- *max_len* maximum buffer size

Returns: The usbredirection at this iterator location.

getName

Description: Gets the Text redirection name

Syntax: void (*getName) (DSDKUSBRedirection* usbr, char* str, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *str* The name string
- *max_len* maximum buffer size

getCreationClassName

Description: Gets CreationClassName

Syntax: void (*getCreationClassName) (DSDKUSBRedirection* usbr, char* str, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *str* The name string
- *max_len* maximum buffer size

Returns: The creationclassname

getElementName

Description: Gets ElementName

Syntax: void (*getElementName) (DSDKUSBRedirection* usbr, char* str, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *str* The name string
- *max_len* maximum buffer size

Returns: The ElementName

getEnabledState

Description: Gets the state of the usb redirection

Syntax: uint16 (*getEnabledState) (DSDKUSBRedirection* usbr);

Parameters:

- *usbr* usbredirection iterator.

Returns: The enabled state

getEnabledStateStr

Description: Gets the state of the usb redirection as string

Syntax: void (*getEnabledStateStr) (DSDKUSBRedirection* usbr, char* state, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *state* The Enabled State is filled in
- *max_len* maximum buffer size

getRequestedState

Description: Gets the requested state of the usb redirection

Syntax: uint16 (*getRequestedState) (DSDKUSBRedirection* usbr);

Parameters:

- *usbr* pointer to usbredirection.

Returns: The enabled state

getRequestedStateStr

Description: Gets the requested status of the usb redirection as string

Syntax: void (*getRequestedStateStr) (DSDKUSBRedirection* usbr, char* state, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *state* The Enabled State is filled in
- *max_len* maximum buffer size

getConnectionMode

Description: Gets the connection mode

Syntax: void (*getConnectionMode) (DSDKUSBRedirection* usbr, char* cmode, int max_len);

Parameters:

- *usbr* pointer to usbredirection.
- *cmode* The connection modem
- *max_len* maximum buffer size

activate

Description: Enable/Activate this redirection session
Syntax: void (*activate) (DSDKUSBRedirection* usbr);
Parameters:
- *usbr* pointer to usbredirection.

enable

Description: Enable this redirection session.
Syntax: void (*enable) (DSDKUSBRedirection* usbr);
Parameters:
- *usbr* pointer to usbredirection.

disable

Description: Disable this redirection session
Syntax: void (*disable) (DSDKUSBRedirection* usbr);
Parameters:
- *usbr* pointer to usbredirection.

release

Description: Releases this object
Syntax: void(* release)(DSDKUSBRedirectionIterator *di);
Parameters:
- *di* pointer to usbredirection.

DSDKUSBRedirection

A structure representing a usbredirection.

Structure Members

hdl - Opaque pointer to usbredirection specific implementations
ft - Pointer to usbredirection function table.

DSDKUSBRedirectionFT

A structure representing usbredirection function table.

Member Functions

getName
getConnectionMode
activate
deactivate
disable
release

Member Function Descriptions

getName

Description: Gets the name of usbredirection

Syntax: void (*getName)(DSDKUSBRedirection *usbr,**char * name, int max_len**);

Parameters:

- *usbr* pointer to usbredirection.

Returns: The name of usbredirection

getConnectionMode

Description: Gets the text flow type of usbredirection

Syntax: void (*getConnectionMode)(DSDKUSBRedirection usbr,char *cmode, int max_len);

Parameters:

- *usbr* pointer to usbredirection.

Returns: The connection mode of usbredirection.

activate

Description: Activate of usbredirection

Syntax: void (*activate)(DSDKUSBRedirection *usbr);

Parameters:

- *usbr* pointer to usbredirection.

Returns : The activate of usbredirection

deactivate

Description: Deactivate of usbredirection

Syntax: void (*deactivate)(DSDKUSBRedirection *usbr);

Parameters:

- *usbr* pointer to usbredirection.

Returns : The deactivate of usbredirection

disable

Description: Disable of usbredirection

Syntax: void (*disable)(DSDKUSBRedirection *usbr);

Parameters:

- *usbr* pointer to usbredirection.

Returns : The disable of usbredirection.

release

Description: Releases this object

Syntax: void(* release)(DSDKUSBRedirection *usbr);

Parameters:

- *usbr* Pointer to DSDKUSBRedirection

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- usbredirection_c.h
Library -- dashapic

5.2.24. VirtualMedia

DSDKVirtualMedia

A structure representing a virtualmedia

Structure Members

hdl - Opaque pointer to virtualmedia specific implementations
ft - Pointer to virtualmedia function table.

DSDKVirtualMedia

A structure representing a virtualmedia.

Structure Members

hdl - Opaque pointer to virtualmedia specific implementations
ft - Pointer to virtualmedia function table.

MakeVirtualMedia

Syntax: DSDKC_Export DSDKVirtualMedia* makeVirtualMedia (char* host, int port, char* user,char* password, char* name, RedirectionType_E type, int is_secure,int is_write_support);

getLocalDrives

Syntax : int getLocalDrives (char** drives, int max_len);

DSDKVirtualMediaFT

A structure representing virtualmedia function table.

Member Functions

startRedirection

stopRedirection
getLocalDrives
release

Member Function Descriptions

startRedirection

Description: starts redirection of virtualmedia
Syntax: int (*startRedirection)(DSDKVirtualMedia *vm);
Parameters:
- *vm* Pointer to virtualmedia

Returns: starts redirection

stopRedirection

Description: stop redirection of virtualmedia
Syntax: int (*stopRedirection)(DSDKVirtualMedia *vm);
Parameters:
- *vm* Pointer to virtualmedia

Returns: stop redirection

release

Description: Releases this object
Syntax: void(* release)(DSDKVirtualMedia *vm);
Parameters:
- *vm* Pointer to DSDKVirtualMedia

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- virtualmedia_c.h
Library -- dashapic

5.2.25. EthernetPort

DSDKEthernetPortIterator

A structure representing a EthernetPort iterator.

Structure Members

hdl - Opaque pointer to EthernetPort specific implementations
ft - Pointer to EthernetPort iterator function table.

enumEthernetPorts

Description: Enumerate all the EthernetPort present under a management access point.

Syntax: DSDKEthernetPortIterator* enumEthernetPorts (DSDKClient* client, BOOL cached)

Parameters:

- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The EthernetPort iterator.

E**thernetPortsIteratorFt**

A structure representing EthernetPorts iterator function table.

Member Functions

getItem
isEnd
next
release

Member Function Descriptions

getItem

Description: Gets the EthernetPorts at this iterator location.

Syntax: DSDKEthernetPort* ethi_getItem (DSDKEthernetPortIterator* di)

Parameters:

- *di* EthernetPort iterator.

Returns: The EthernetPort at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.

Syntax: BOOL(* isEnd)(DSDKEthernetPortIterator *di);

Parameters:

- *di* EthernetPort iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKEthernetPortIterator *di);

Parameters:

- *di* EthernetPort iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKEthernetPortIterator *di);

Parameters:

- *di* EthernetPort iterator.

DSDKEthernetPort

A structure representing a EthernetPort

Structure Members

hdl - Opaque pointer to EthernetPort specific implementations

ft - Pointer to EthernetPort function table.

DSDKEthernetPort FT

A structure representing EthernetPort function table.

Member Functions

getPortType
getPortTypeStr
getNetworkAddresses
getCapabilities
getCapabilitiesStr
getEnabledCapabilities
getEnabledCapabilitiesStr
getLinkTechnology
getPermanentAddress
getDeviceID
release

Member Function Descriptions

getPortType

Description: Gets Port Type

Syntax: uint16 (*getPortType) (DSDKEthernetPort* eth);

Parameters:

- *eth* pointer to EthernetPort

Returns: Port Type

getPortTypeStr

Description: Gets port type as string

Syntax: void (*getPortTypeStr) (DSDKEthernetPort* eth, char* type, int max_len);

Parameters:

- *eth* pointer to EthernetPort
- *type* The Port Type
- *max_len* maximum buffer length

getNetworkAddresses

Description: Gets an array of NetworkAddresses

Syntax: int (*getNetworkAddresses) (DSDKEthernetPort* eth, char** na, int max_na, int max_strlen);

Parameters:

- *eth* pointer to EthernetPort
- *na* network addresses
- *max_na* maximum network address
- *max_strlen* maximum buffer length

getCapabilities

Description: Gets the Capabilities

Syntax: int (*getCapabilities) (DSDKEthernetPort* eth, uint16* capabilities, int max_types);

Parameters:

- *eth* pointer to EthernetPort

capabilities Capabilities

max_types maximum types

getCapabilitiesStr

Description: Gets capabilities as string

Syntax: int (*getCapabilitiesStr) (DSDKEthernetPort* eth, char** capabilities, int max_capabilities, int max_strlen);

Parameters:

- *eth* pointer to EthernetPort
- *capabilities* The capabilities
- *max_capabilities* maximum capabisities
- *mmax_strlen* maximum string length

getEnabledCapabilities

Description: Gets the EnabledCapabilities

Syntax: int (*getEnabledCapabilities) (DSDKEthernetPort* eth, uint16* enab_capabilities, int max_types);

Parameters:

- *eth* pointer to EthernetPort
- *enab_capabilities* Enabled Capabilities
- *max_types* maximum types

getEnabledCapabilitiesStr

Description: Gets enabled capabilities as string

Syntax: int (*getEnabledCapabilitiesStr) (DSDKEthernetPort* eth, char** capabilities, int max_capabilities, int max_strlen);

Parameters:

- *eth* pointer to EthernetPort
- *capabilities* The capabilities
- *max_capabilities* maximum capabilitises
- *mmax_strlen* maximum string length

getLinkTechnology

Description: Gets getLinkTechnology

Syntax: uint16 (*getLinkTechnology) (DSDKEthernetPort* eth);

Parameters:

- *eth* pointer to EthernetPort

Returns: The Link Technology

getPermanentAddress

Description: Gets Permanant address

Syntax: void (*getPermanentAddress) (DSDKEthernetPort* eth, char* perm_addr, int max_len);

Parameters:

- *eth* pointer to EthernetPort
- *perm_addr* The Permanant address
- *max_len* maximum buffer length

getDeviceID

Description: Gets Device ID

Syntax: void (*getDeviceID) (DSDKEthernetPort* eth, char* dev_id, int max_len);

Parameters:

- *eth* pointer to EthernetPort
- *dev_id* The Device ID
- *max_len* maximum buffer length

release

Description: Releases this object

Syntax: void (*release) (DSDKEthernetPort* eth);

Parameters:

- *eth* pointer to EthernetPort

Note: All functions in C library should call *dsdkc_getLastError* function to get the status of API function execution. If getLastError returns 0, API function is executed successfully. If non zero is returned then use *dsdkc_getLastErrorStr* to get the error description. These functions are explained at end of this section.

Class Requirements

Header file -- ethernetport_c.h
Library -- dashapic

5.2.26. RegisteredProfile

DSDKRegisteredProfileIterator

A structure representing a RegisteredProfile iterator.

Structure Members

hdl - Opaque pointer to RegisteredProfile specific implementations
ft - Pointer to RegisteredProfile iterator function table.

enumRegisteredProfile

Description: Enumerate all the RegisteredProfile present under a management access point.
Syntax: DSDKRegisteredProfileIterator* enumRegisteredProfile (DSDKClient* client, BOOL cached);
Parameters:
- *client* Pointer to the client interface.
- *cached* Enable/Disable caching.

Returns: The RegisteredProfile iterator.

RegisteredProfile IteratorFt

A structure representing RegisteredProfile iterator function table.

Member Functions

getItem
isEnd
next
release

Member Function Descriptions

getItem

Description: Gets the RegisteredProfile at this iterator location.
Syntax: DSDKRegisteredProfile*(*getItem)(DSDKRegisteredProfileIterator *di);
Parameters:
- *di* RegisteredProfile iterator.

Returns: The RegisteredProfile at this iterator location.

isEnd

Description: Returns true if iterator have reached the end.
Syntax: BOOL(* isEnd)(DSDKRegisteredProfileIterator *di);
Parameters:

- *di* RegisteredProfile iterator.

Returns: True if have reached the end.

next

Description: Moves the iterator to the next location.

Syntax: void(* next)(DSDKRegisteredProfileIterator *di);

Parameters:

- *di* RegisteredProfile iterator.

release

Description: Releases this object

Syntax: void(* release)(DSDKRegisteredProfileIterator *di);

Parameters:

- *di* RegisteredProfile iterator.

DSDKRegisteredProfile

A structure representing a RegisteredProfile

Structure Members

hdl - Opaque pointer to RegisteredProfile specific implementations

ft - Pointer to RegisteredProfile function table.

DSDKRegisteredProfileFT

A structure representing RegisteredProfile function table.

Member Functions

getAdvertiseTypes
getInstanceID
getRegisteredName
getRegisteredOrganization
getValueRegisteredOrganizationStr
getRegisteredVersion
release

Member Function Descriptions

getAdvertiseTypes

Description: Gets the AdvertiseTypes

Syntax: int (*getAdvertiseTypes) (DSDKRegisteredProfile* rp, uint16* adv_types, int max_types);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns: The AdvertiseTypes

getInstanceID

Description: Gets the InstanceID

Syntax: void (*getInstanceID) (DSDKRegisteredProfile* rp, char* insid, int max_len);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns: The InstanceID

getRegisteredName

Description: Gets the RegisteredName

Syntax: void (*getRegisteredName) (DSDKRegisteredProfile* rp, char* reg_name, int max_len);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns: The RegisteredName

getRegisteredOrganization

Description: Gets the RegisteredOrganization

Syntax: uint64 (*getRegisteredOrganization) (DSDKRegisteredProfile* rp);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns: The RegisteredOrganization

getValueRegisteredOrganizationStr

Description: Gets the ValueRegisteredOrganization as string

Syntax: void (*getValueRegisteredOrganizationStr) (DSDKRegisteredProfile* rp, char* regorgstr, int max_len);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns: The ValueRegisteredOrganization as string

getRegisteredVersion

Description: Gets the RegisteredVersion

Syntax: void (*getRegisteredVersion) (DSDKRegisteredProfile* rp, char* reg_version, int max_len);

Parameters:

- *rp* Pointer to RegisteredProfile

Returns**:** The RegisteredVersion

release

Description: Releases this object

Syntax: void(* release)(DSDKRegisteredProfile *rp);

Parameters:

- *rp* Pointer to __DSDKRegisteredProfile

5.2.27. Error functions

Error functions are used to get the error status of API function execution.
dsdk_getLastError
dsdk_getLastErrStr

dsdk_getLastError

Description**:** Gets error code for last executed C API function.
Syntax: void dsdkc_getLastError (void);
Returns**:** Returns the error code for last executed function.

dsdk_getLastErrStr

Description**:** Gets error description for last error code.
Syntax: void dsdkc_getLastErrorStr (char* str, int max_len);
Parameters:

- *str* Pointer to the buffer that receives the error description.
- *max_len* Length of the buffer.

6 Low Level API

The Low Level API is defined to match the CMPI interface. In addition to this, functions are added to provide support for discovery, DASH ping and eventing. The Low level API is a pure C interface. The following is the list of Low level API functions.

6.1 CMCIClient

A structure representing a Low-Level CMCI client.

Structure Members

hdl - Opaque pointer to CMCI specific implementations
ft - Pointer to CMCI client function table.

6.2 cmciConnect

This function creates and initializes the cmci runtime enviornment , and on successful initialization it returns a pointer to the CMCI Client and null on failure.

Syntax: CMCIClient* cmciConnect (const char * host_name,
const char* scheme,
const char* port,
const char* user,

```
                                                const char* pwd,
                                                const char* auth,
                                                CMPIStatus * rc);
```

Parameters:

host_name	Name of the server to connect.
Scheme	HTTP scheme(http/https).
Port	Port number
user	user name
pwd	password
auth	Authentication type.
rc	Status.

This function retuns CMCI client pointer on success, NULL on failure. The CMCI client pointer shouled be released using release function when CMCI client is no longer needed.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient *cc;
        CMPIStatus status;

        /* Setup a conncetion to the CIMOM */
        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin", "admin",
                                                "basic", NULL);


        .....
        ..

        CMRelease (cc);
        return 0;
}
```

6.3 CMCIClientFT

A structure representing CMCI client function table.

Member functions

This structure has following functions as listed below.

enumInstanceNames
enumInstances
getInstance
setInstance
createInstance
deleteInstance
execQuery
associators
associatorNames
references
referenceNames
invokeMethod
setProperty
getProperty
subscribeEvent
unsubscribeEvent
renewSubscription

Member function description.

The following section provides more details on the class member functions

enumInstanceNames

This function is used to enumerate the Objectpaths.

Syntax: CMPIEnumeration* enumInstanceNames(CMCIClient * mb,
CMPIObjectPath * cop,
CMPIStatus * rc);

Parameters:

- *mb* CMCI client pointer returned from cmciConnect.
- *cop* Object path containing name space and class name and key components.
- *rc* CMPI Status, contains the error status.

This function returns the enumeration of object paths.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
```

```
        CMCIClient *cc;
        CMPIStatus status;
        CMPIObjectPath*         objectpath;
        CMPIEnumeration* enumeration;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                                "admin", "basic", NULL);

        objectpath        = newCMPIObjectPath ("root/cimv2",
                                                "CIM_ComputerSystem", NULL);
        enumeration = cc->ft->enumInstanceNames (cc, objectpath, &status);
        if (!status.rc)
        {
                while (enumeration->ft->hasNext (enumeration, NULL))
                {
                        CMPIData data = enumeration->ft->getNext (enumeration, NULL);

                        /* data.value.ref will contain the reference to enumerated
        objects */
                }
        }

        if (enumeration) CMRelease (enumeration);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);

        CMRelease (cc);
        return 0;
}
```

enumInstances

This function is used enumerate the instances.

Syntax: CMPIEnumeration enumInstances (CMCIClient * mb,
CMPIObjectPath* cop,
CMPIFlags flags,
char ** properties,
CMPIStatus * rc);

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *flags* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function retuns the enumeration of instances.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient *cc;
        CMPIStatus status;
        CMPIObjectPath*         objectpath;
        CMPIEnumeration* enumeration;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                                "admin", "basic", NULL);

        objectpath      = newCMPIObjectPath ("root/cimv2", "CIM_ComputerSystem", NULL);
        enumeration = cc->ft->enumInstances (cc, objectpath, 0, NULL, &status);

        if (!status.rc)
        {
                while (enumeration->ft->hasNext (enumeration, NULL))
                {
                        CMPIData data = enumeration->ft->getNext (enumeration, NULL);

                        /* data.value.inst will contain the enumeration instances*/
                }
        }

        if (enumeration) CMRelease (enumeration);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);

        CMRelease (cc);
        return 0;
}
```

getInstance

This function gets a instance.

Syntax: CMPIInstance getInstance (CMCIClient * mb,
CMPIObjectPath* cop,
CMPIFlags flags,

```
                                        char** properties,
                                        CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *flags* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function retuns an instance.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient              *cc;
        CMPIStatus              status;
        CMPIObjectPath          *objectpath;
        CMPIInstance   *instance;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                        "admin", "basic", NULL);

        objectpath      = newCMPIObjectPath ("root/cimv2",
          "http://schemas.dri.org/wbem/wscim/1/cim-schema/2/DRI_ComputerSystem",
                                                NULL);
        CMAddKey (objectpath,  "CreationClassName", "DRI_ComputerSystem",
                                                        CMPI_chars);
        CMAddKey (objectpath,  "Name", "mkl-desktop", CMPI_chars);

        instance = cc->ft->getInstance (cc, objectpath, 0, NULL, &status);

        if (instance)    CMRelease (instance);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg)  CMRelease (status.msg);

        CMRelease (cc);
        return 0;
}
```

setInstance

This function sets an instance.

Syntax: CMPIStatus setInstance (CMCIClient * mb,
CMPIObjectPath* cop,
CMPIInstance* inst,
CMPIFlags flags,
char ** properties)

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *Inst* Instance to set.
- *flags* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function retuns the status

Example:

Below is an example on how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient*             cc;
        CMPIStatus              status;
        CMPIObjectPath*         objectpath;
        CMPIInstance*  instance;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                        "admin", "basic", NULL);

        objectpath      = newCMPIObjectPath("root/cimv2",
                          "http://schemas.dri.org/wbem/wscim/1/cim-schema/2/DRI_Account",
                                                        NULL);
        CMAddKey (objectpath,  "Name", "Thomas", CMPI_chars);
        CMAddKey (objectpath,  "CreationClassName", "DRI_Account", CMPI_chars);
        CMAddKey (objectpath,  "SystemName", "mkl-desktop", CMPI_chars);
        CMAddKey (objectpath,  "SystemCreationClassName", "DRI_ComputerSystem",
```

```
                CMPI_chars);

        instance        = newCMPIInstance(objectpath, NULL);
        CMSetProperty (instance, "Caption", "Account modified", CMPI_chars);

        status = cc->ft->setInstance (cc, objectpath, instance, 0, NULL);

        if (instance)    CMRelease (instance);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);
        if (cc) CMRelease(cc);

        CMRelease (cc);
        return 0;
}
```

createInstance

This function creates a new instance specified by inst.

Syntax: CMPIObjectPath* createInstance (CMCIClient * mb,
CMPIObjectPath* cop,
CMPIInstance* inst,
CMPIStatus* rc);

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *Inst* Instance to create
- *rc* CMPI Status, contains the error status.

This function retuns the object path of newly created instance.

deleteInstance

This function delets an instance.

Syntax: CMPIStatus deleteInstance (CMCIClient * mb,
CMPIObjectPath * cop);

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components of the instance to delete

This function returns the status of command.

execQuery

This function is used to enumerate the instance based on query language.

Syntax: CMPIEnumeration* execQuery (CMCIClient * mb,
CMPIObjectPath * cop,
const char * query,
const char * lang,
CMPIStatus * rc)

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *query* Query expression
- *lang* Query language
- *rc* CMPI Status, contains the error status.

This function returns the resulting enumeration of instances.

Example:

Below is an example on how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient*             cc;
        CMPIStatus              status;
        CMPIObjectPath*         objectpath;
        CMPIEnumeration*        enumeration;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                        "admin", "basic", NULL);

        objectpath      = newCMPIObjectPath ("root/cimv2",
                "http://schemas.dri.org/wbem/wscim/1/cim-schema/2/DRI_ComputerSystem",
                                                        NULL);

        enumeration = cc->ft->execQuery (cc, objectpath,
                                        "SELECT * FROM CIM_ComputerSystem WHERE
                                        CreationClassname=\"DRI_ComputerSystem\"",
                                        "WQL",
                                        &status);

        if (!status.rc)
        {
                while (enumeration->ft->hasNext (enumeration, NULL))
                {
```

```
                        CMPIData data = enumeration->ft->getNext (enumeration, NULL);
                }
        }

        if (enumeration) CMRelease (enumeration);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);

        CMRelease (cc);
        return 0;
}
```

associators

This function is used to enumerate instances that are associated with the instance pointed the object path cop.

Syntax:

```
CMPIEnumeration* associators (CMCIClient * mb,
                              CMPIObjectPath * cop,
                              const char* assocClass,
                              const char* resultClass,
                              const char* role,
                              const char* resultRole,
                              CMPIFlags flags,
                              char ** properties,
                              CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *assocClass* Association class.
- *resultClass* Result Class.
- *role* Role.
- *resultRole* Result Role
- *flag* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function returns the resulting enumeration of instances.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"
```

```
int main ( int argc, char * argv[] )
{
        CMCIClient*             cc;
        CMPIStatus              status;
        CMPIObjectPath*         objectpath;
        CMPIEnumeration*        enumeration;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                                "admin", "basic", NULL);

        objectpath = newCMPIObjectPath ("root/cimv2",
                "http://schemas.dri.org/wbem/wscim/1/cim-schema/2/DRI_ComputerSystem",
                                                        NULL);

        CMAddKey (objectpath, "CreationClassName", "DRI_ComputerSystem", CMPI_chars);
        CMAddKey (objectpath, "Name", "mkl-desktop", CMPI_chars);

        enumeration = cc->ft->associators  (cc, objectpath,
                                "CIM_SystemDevice",
                                        NULL, NULL, NULL, 0, NULL, &status);

        if (!status.rc)
        {
                while (enumeration->ft->hasNext (enumeration, NULL))
                {
                CMPIData data = enumeration->ft->getNext (enumeration, NULL);

                }
        }

        if (enumeration) CMRelease (enumeration);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);


        CMRelease (cc);
        return 0;
}
```

associatorNames

This function is used to enumerate the instances that are associated with the instance pointed to the object path cop.

Syntax:

```
CMPIEnumeration* associatorsNames (CMCIClient * mb,
                                   CMPIObjectPath * cop,
                                   const char* assocClass,
                                   const char*
                                   resultClass,
                                   const char* role,
```

```
                                        const char* resultRole,
                                        CMPIFlags flags,
                                        char ** properties,
                                        CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *assocClass* Association class.
- *resultClass* Result Class.
- *role* Role.
- *resultRole* Result Role
- *flag* This flag is not used, and for backward compatibility with sfcc its not removed.
- *oroperties* property names.
- *rc* CMPI Status, contains the error status.

This function returns the resulting enumeration of objectpaths.

references

This function is used to enumerate the instances that refer the instance pointed by object path cop.

Syntax:

```
CMPIEnumeration references (CMCIClient * mb,
                            CMPIObjectPath * cop,
                            const char* resultClass,
                            const char* role,
                            CMPIFlags flags,
                            char** properties,
                            CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *resultClass* Result Class.
- *role* Role.
- *flag* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function returns the resulting enumeration of instances.

Example:

Below is an example of how to use this function.

```
#include "cmci.h"
#include "test.h"

int main ( int argc, char * argv[] )
{
        CMCIClient*                     cc;
        CMPIStatus                      status;
        CMPIObjectPath*                 objectpath;
        CMPIEnumeration*        enumeration;

        cc = cmciConnect ("192.168.0.20", NULL, "623", "admin",
                                                                        "admin", "basic", NULL);

        objectpath = newCMPIObjectPath ("root/cimv2",
                "http://schemas.dri.org/wbem/wscim/1/cim-schema/2/DRI_ComputerSystem",
                                                        NULL);
        CMAddKey (objectpath, "CreationClassName", "DRI_ComputerSystem",
                                                                        CMPI_chars);
        CMAddKey (objectpath, "Name", "mkl-desktop", CMPI_chars);


        enumeration = cc->ft->references (cc, objectpath, NULL, NULL, 0, NULL,
                                                                        &status);

        if (!status.rc)
        {
                while (enumeration->ft->hasNext (enumeration, NULL))
                {
                        CMPIData data = enumeration->ft->getNext (enumeration, NULL);
                }
        }

        if (enumeration) CMRelease (enumeration);
        if (objectpath)  CMRelease (objectpath);
        if (status.msg) CMRelease (status.msg);

        CMRelease (cc);
        return 0;
}
```

referenceNames

This function is used to enumerate the instances that refer the instance pointed by object path cop.

Syntax: CMPIEnumeration referenceNames (CMCIClient * mb,
CMPIObjectPath * cop,
const char* resultClass,
const char* role,
CMPIFlags flags,

```
                                                    char** properties,
                                                     CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *resultClass* Result Class.
- *role* Role.
- *flag* This flag is not used, and for backward compatibility with sfcc its not removed.
- *properties* property names.
- *rc* CMPI Status, contains the error status.

This function returns the resulting enumeration of object paths.

invokeMethod

This function is used to invoke a method

Syntax:

```
CMPIData invokeMethod (CMCIClient * mb, CMPIObjectPath *
                                                    cop, const char * method,
                                                    CMPIArgs * in,
                                                    CMPIArgs * out,
                                                    CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *method* The name of the method to invoke.
- *in* Input argument to the method.
- *out* Output argument from the method
- *rc* CMPI Status, contains the error status.

This function returns a Method return value.

setProperty

This function sets a property of instance.

Syntax:

```
CMPIStatus setProperty (CMCIClient * mb,
                                                    CMPIObjectPath* cop,
                                                     const char* name,
                                                    CMPIValue* value,
                                                    CMPIType type);
```

Parameters:

- *mb* CMCI client pointer

- *cop* Object path containing name space and class name and key components.
- *name* The name of the property.
- *value* Value to set.
- *type* Type of the property.

This function retuns the status.

getProperty

This function gets the property value of instance.

Syntax:
```
CMPIData getProperty (CMCIClient * mb,
                      CMPIObjectPath * cop,
                      const * char name,
                      CMPIStatus * rc);
```

Parameters:

- *mb* CMCI client pointer
- *cop* Object path containing name space and class name and key components.
- *name* The name of the property.
- *rc* CMPI Status

This function returns the property value.

subscribeEvent

This function subscribes for an event.

Syntax:
```
char* subscribeEvent (CMCIClient * mb,
                      const char * event_uri,
                      int mode,
                      float heartbeat,
                      float expires,
                      const char* dialect,
                      const char* filter,
                      const char* resourceUri,
                      CMPIStatus * rc)
```

Parameters:

- *mb* CMCI client pointer
- *event_uri* Destination uri where the events to sent.
- *hearbeat* Heartbeat interval in seconds
- *expires* Expire time for the event to expire.
- *dialect* dialect filter.
- *filter* event filter.
- *resource_ur* Resource URI.

This functions returns the subscription ID.

unsubscribeEvent

Un subscribes an event

Syntax: CMPIStatus unsubscribeEvent (CMCIClient * mb, const char * uuid);

Parameters:

- *mb* CMCI client pointer
- *uuid* The event UUDI to unsubscribe.

renewSubscription

This function renews an event subscription

Syntax: CMPIStatus renewSubscription (CMCIClient * mb, const char * uuid);

Parameters:

- *mb* CMCI client pointer
- *uuid* The event UUID to unsubscribe.