



Cisco Active Network Abstraction BQL User's Guide, 3.5

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100



THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCSP, CCVP, the Cisco Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Access Registrar, Aironet, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, *Packet*, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, ScriptShare, SlideCast, SMARTnet, The Fastest Way to Increase Your Internet Quotient, and TransPath are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0601R)

Important Notice

Cisco ANA 3.5 is a carrier-class, multi-vendor network and service management platform which builds a real-time virtual model of the network, serving as a live information base for value-added tools and applications for integration into an existing OSS environment.

Cisco ANA 3.5 is a limited release by Cisco Systems of the existing features and functions of the Sheer DNA 4.0.1 software.

As this is a limited release, the naming of the product in the software and the user documentation remains as Sheer DNA.

Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation at this URL:

<http://www.cisco.com/techsupport>

You can access the Cisco website at this URL:

<http://www.cisco.com>

You can access international Cisco websites at this URL:

http://www.cisco.com/public/countries_languages.shtml

Product Documentation DVD

Cisco documentation and additional literature are available in the Product Documentation DVD package, which may have shipped with your product. The Product Documentation DVD is updated regularly and may be more current than printed documentation.

The Product Documentation DVD is a comprehensive library of technical product documentation on portable media. The DVD enables you to access multiple versions of hardware and software installation, configuration, and command guides for Cisco products and to view technical documentation in HTML. With the DVD, you have access to the same documentation that is found on the Cisco website without being connected to the Internet. Certain products also have .pdf versions of the documentation available.

The Product Documentation DVD is available as a single unit or as a subscription. Registered Cisco.com users (Cisco direct customers) can order a Product Documentation DVD (product number DOC-DOCDVD=) from Cisco Marketplace at this URL:

<http://www.cisco.com/go/marketplace/>

Ordering Documentation

Beginning June 30, 2005, registered Cisco.com users may order Cisco documentation at the Product Documentation Store in the Cisco Marketplace at this URL:

<http://www.cisco.com/go/marketplace/>

Nonregistered Cisco.com users can order technical documentation from 8:00 a.m. to 5:00 p.m. (0800 to 1700) PDT by calling 1 866 463-3487 in the United States and Canada, or elsewhere by calling 011 408 519-5055. You can also order documentation by e-mail at tech-doc-store-mkpl@external.cisco.com or by fax at 1 408 519-5001 in the United States and Canada, or elsewhere at 011 408 519-5001.

Documentation Feedback

You can rate and provide feedback about Cisco technical documents by completing the online feedback form that appears with the technical documents on Cisco.com.

You can send comments about Cisco documentation to bug-doc@cisco.com.

You can submit comments by using the response card (if present) behind the front cover of your document or by writing to the following address:

Cisco Systems
Attn: Customer Document Ordering
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Cisco Product Security Overview

Cisco provides a free online Security Vulnerability Policy portal at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

From this site, you can perform these tasks:

- Report security vulnerabilities in Cisco products.
- Obtain assistance with security incidents that involve Cisco products.
- Register to receive security information from Cisco.

A current list of security advisories and notices for Cisco products is available at this URL:

<http://www.cisco.com/go/psirt>

If you prefer to see advisories and notices as they are updated in real time, you can access a Product Security Incident Response Team Really Simple Syndication (PSIRT RSS) feed from this URL:

http://www.cisco.com/en/US/products/products_psirt_rss_feed.html

Reporting Security Problems in Cisco Products

Cisco is committed to delivering secure products. We test our products internally before we release them, and we strive to correct all vulnerabilities quickly. If you think that you might have identified a vulnerability in a Cisco product, contact PSIRT:

Emergencies — security-alert@cisco.com

An emergency is either a condition in which a system is under active attack or a condition for which a severe and urgent security vulnerability should be reported. All other conditions are considered nonemergencies.

Nonemergencies — psirt@cisco.com

In an emergency, you can also reach PSIRT by telephone:

1 877 228-7302

1 408 525-6532

We encourage you to use Pretty Good Privacy (PGP) or a compatible product to encrypt any sensitive information that you send to Cisco. PSIRT can work from encrypted information that is compatible with PGP versions 2.x through 8.x.

Never use a revoked or an expired encryption key. The correct public key to use in your correspondence with PSIRT is the one linked in the Contact Summary section of the Security Vulnerability Policy page at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

The link on this page has the current PGP key ID in use.

Obtaining Technical Assistance

Cisco Technical Support provides 24-hour-a-day award-winning technical assistance. The Cisco Technical Support & Documentation website on Cisco.com features extensive online support resources. In addition, if you have a valid Cisco service contract, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not have a valid Cisco service contract, contact your reseller.

Cisco Technical Support & Documentation Website

The Cisco Technical Support & Documentation website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day, at this URL:

<http://www.cisco.com/techsupport>

Access to all tools on the Cisco Technical Support & Documentation website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

<http://tools.cisco.com/RPF/register/register.do>

Use the Cisco Product Identification (CPI) tool to locate your product serial number before submitting a web or phone request for service. You can access the CPI tool from the Cisco Technical Support & Documentation website by clicking the Tools & Resources link under Documentation & Tools. Choose Cisco Product Identification Tool from the Alphabetical Index drop-down list, or click the Cisco Product Identification Tool link under Alerts & RMAs. The CPI tool offers three search options: by product ID or model name; by tree view; or for certain products, by copying and pasting show command output. Search results show an illustration of your product with the serial number label location highlighted. Locate the serial number label on your product and record the information before placing a service call.

Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool provides recommended solutions. If your issue is not resolved using the recommended resources, your service request is assigned to a Cisco engineer. The TAC Service Request Tool is located at this URL:

<http://www.cisco.com/techsupport/servicerequest>

For S1 or S2 service requests or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

- Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)
- EMEA: +32 2 704 55 55
- USA: 1 800 553-2447

For a complete list of Cisco TAC contacts, go to this URL:

<http://www.cisco.com/techsupport/contacts>

Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

- Severity 1 (S1)—Your network is “down,” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.
- Severity 2 (S2)—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.
- Severity 3 (S3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.
- Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

Cisco Marketplace provides a variety of Cisco books, reference guides, documentation, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

<http://www.cisco.com/go/marketplace/>

Cisco Press publishes a wide range of general networking, training and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:

<http://www.ciscopress.com>

Packet magazine is the Cisco Systems technical user magazine for maximizing Internet and networking investments. Each quarter, Packet delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can access Packet magazine at this URL:

<http://www.cisco.com/packet>

iQ Magazine is the quarterly publication from Cisco Systems designed to help growing companies learn how they can use technology to increase revenue, streamline their business, and expand services. The publication identifies the challenges facing these companies and the technologies to help solve them, using real-world case studies and business strategies to help readers make sound technology investment decisions. You can access iQ Magazine at this URL:

<http://www.cisco.com/go/iqmagazine>

or view the digital edition at this URL:

<http://cisoiq.texterity.com/cisoiq/sample/>

Internet Protocol Journal is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

<http://www.cisco.com/ipj>

Networking products offered by Cisco Systems, as well as customer support services, can be obtained at this URL:

<http://www.cisco.com/en/US/products/index.html>

Networking Professionals Connection is an interactive website for networking professionals to share questions, suggestions, and information about networking products and technologies with Cisco experts and other networking professionals. Join a discussion at this URL:

<http://www.cisco.com/discuss/networking>

World-class networking training is available from Cisco. You can view current offerings at this URL:

<http://www.cisco.com/en/US/learning/index.html>

About This Guide

Objectives

This document is a basic user guide for Sheer Networks BQL (*Broadband Query Language*), an open XML-based query language for northbound integration with the Sheer Networks DNA (*Dynamic Network Abstraction*) Network & Service management platform.

The guide describes the basic functions and concepts of BQL and IMO (*Information Model Objects* – Sheer’s internal information model), and provides basic BQL code examples. The user manual should be accompanied with the *IMO reference manual* and/or *IMO reference browser* for detailed IMO reference.

Audience

The intended audience for this document is system architects and programmers who wish to access DNA and integrate it with northbound OSS applications. The document assumes the reader is familiar with XML and with application programming.

Document Organization

The document provides gradual introduction of the user into BQL and IMO. The first section enables the user to “Get started” by describing the basic BQL/IMO concepts, and providing some simple programming examples. The next sections describe the advanced features of IMO and BQL in further details.

Table of Contents

1	Getting Started	1
1.1	What is BQL?	1
1.2	Quick Start - BQL Basics	1
1.2.1	Connecting to BQL	1
1.2.2	BQL Commands	3
1.2.3	Device-List Report Example	4
1.2.4	The BQL Get Command	5
2	Understanding IMO.....	9
2.1	What is IMO?	9
2.2	IMO Basics	9
2.3	De-referencing.....	10
2.4	IMO by Example.....	12
2.4.1	Simple IMO object Example	12
2.4.2	IMO Construct Example	13
2.5	Typical IMO Constructs.....	15
2.5.1	Inventory Containment	15
2.5.2	Port Connections	15
2.5.3	Service Paths (SNCs).....	16
3	IMO Specification.....	21
3.1	IMO Format/Syntax.....	21
3.2	IMO OIDs.....	22
3.2.1	Format/Syntax	22
3.2.2	OID Example.....	23
3.2.3	Sample OID Hierarchies	23
3.3	IMO Primitives	26
3.4	IMO Constructs	28
3.4.1	Nesting IMO Objects.....	28
3.4.2	IMO Object Arrays.....	29
3.4.3	Complex Constructs	29
3.5	Retrieval Specifications	30
3.5.1	Defining the Retrieval Scope.....	30
3.5.2	Format/Syntax	36
3.6	Aspects.....	37

3.6.1	Example	38
3.6.2	Specifying Properties vs. Aspects	39
3.6.3	Required Aspects vs. Excluded Aspects	39
3.7	Registrations and Notifications.....	40
3.7.1	Registering for Notifications	40
3.7.2	Unregistering	41
3.7.3	Notifications	41
4	BQL Generic Commands.....	47
4.1	Introduction.....	47
4.2	GET Command.....	47
4.2.1	Description	47
4.2.2	Syntax	48
4.2.3	Output	48
4.2.4	Example	48
4.3	UPDATE Command.....	49
4.3.1	Description	49
4.3.2	Syntax	49
4.3.3	Output	49
4.3.4	Example	50
4.4	CREATE Command	51
4.4.1	Description	51
4.4.2	Syntax	51
4.4.3	Output	51
4.4.4	Example	51
4.5	DELETE Command.....	52
4.5.1	Description	52
4.5.2	Syntax	52
4.5.3	Output	52
4.5.4	Example	52
4.6	Find	52
4.6.1	Description	52
4.6.2	Syntax	53
4.6.3	Output	53
4.6.4	Example	54

4.7	Refresh	54
4.7.1	Description	54
4.7.2	Example	55

Table of Figures

Figure 1: Integration Sequence	8
Figure 2: IMO Modeling for Inventory Containment	15
Figure 3: IMO Port Modeling	16
Figure 4: IMO Modeling for SNC.....	17
Figure 5: DSL/ATM Service Path	17
Figure 6: Cross Connect Sequence.....	18
Figure 7: IMO Representation for SNC	18
Figure 8: OID Hierarchy.....	23
Figure 9: Alcatel ASAM Port OID.....	24
Figure 10: Lucent CBX Port OID.....	24
Figure 11: RedBack SMS Port OID	25
Figure 12: RedBack SMS Logical OID.....	26

1 Getting Started

1.1 What is BQL?

Sheer BQL (*Broadband Query Language*) is a generic machine interface language, implemented by the Sheer DNA Gateway, for general-purpose northbound integration. BQL covers all of Sheer DNA functionality:

- Information reporting (e.g. inventory, topology, fault etc.)
- Service Activation
- System administration.

The interface is based on XML messages over TCP sockets. All data sent to and returned from the system is formatted as XML messages, containing IMO (Sheer's internal information model) data objects.

1.2 Quick Start - BQL Basics

This section briefly describes the BQL interface and gives an example of how to use it for integration. Each of the topics referenced in this section is elaborated in subsequent sections. The section describes briefly:

- How to connect to the BQL interface
- How to format a command
- How to execute a command
- How to interpret the result of a command
- How to limit the scope of a command
- How to register for data change notifications

The explanations in this section are accompanied with examples S/W programs, written in Perl. However, BQL is not restricted to any specific programming language.

1.2.1 Connecting to BQL

The BQL interface is using a TCP socket transport. The DNA Gateway listens for incoming connection attempts on a well-known port, as follows:

- Protocol: **TCP**
- Port number: **9002**
- Socket type: **Streamable**

In order to open a BQL session, the connecting application must authenticate the session using a username and password (which has been previously defined within the Sheer system). The authentication command is:

```
openconnection user=<Username> password=<Password>
```

Following is a Perl example of opening & authenticating a BQL session, with the DNA Gateway whose IP address is 192.168.2.110:

```
use IO::Socket::INET;

my $sock = IO::Socket::INET->new(PeerAddr => '192.168.2.110',
    PeerPort => 9002,
    Proto => 'tcp',
    Type => SOCK_STREAM);
die "Cannot open socket" unless defined $sock;

print $sock "openconnection user='John' password='XYZ'\n.\n";

my $Recv;
my $LoginOK=0;
while (($Recv=<$sock>)!~ /^\.\/)
{
    if ($Recv =~ /. *success.*\/)
    {
        print "Login OK\n";
        $LoginOK=1;
    }
}
if (!$LoginOK)
{
    print "Login failed\n";
    print $sock "exit\n.\n";
    $sock->close();
    exit;
}
```

Note the `\n.\n` EOT (end-of-text) sequence at the end of the commands, which will be explained in the next section.

To disconnect from the BQL interface, the user has to issue the command:

```
exit\n.\n
```

And then close the connection with the Sheer DNA gateway.

1.2.2 BQL Commands

Once the BQL session is authenticated, all commands and responses are formatted as plain-text XML messages. Each command has the following format:

```
<command name="commandName">
  <param name="paramName1">
    <value>ParamValue1</value>
  </param>
  <param name="paramName2">
    <value>ParamValue2</value>
  </param>
  . . .
</command>
```

Every command must end with an *End-Of-Text (EOT)* sequence, which is a *<NewLine>* character (“\n”) followed by a dot in an empty line, as follows:

EOT: “\n.\n”

Therefore, every command will have the following format:

```
<command name="commandName">
. . .
</command>\n.\n
```

The EOT sequence is also terminating all BQL replies. However, when receiving unsolicited notifications, the EOT sequence changes into the format

Notification EOT: “\n\$\n”

Notifications are further explained in section 3.6.2.

The following example demonstrates a BQL command that retrieves the list of all NEs managed by the system. The command is called *DeviceList* and has no parameters:

```
<command name="DeviceList">
</command>
.
```

After sending the above command to the BQL socket, the socket connection blocks until the Sheer DNA gateway finishes processing the query. Once finished, the Sheer DNA gateway will respond with the list of all devices in the system, and will free the connection for further queries. The reply to the *DeviceList* command will be, for example, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IMManagedElement>
    <ID type="Oid">{[ManagedElement(Key=CBX500Lab)]}</ID>
    <CommunicationStateEnum type="Integer">3</CommunicationStateEnum>
    <DeviceName type="String">CBX500Lab</DeviceName>
    <ElementCategoryEnum type="Integer">2</ElementCategoryEnum>
    <ElementTypeEnum type="Integer">22</ElementTypeEnum>
    <IP type="com.sheer.types.IPAddress">192.168.2.70</IP>
    <InvestigationStateEnum type="Integer">2</InvestigationStateEnum>
```

```

    <SoftwareVersion type="String">04.02.01.00</SoftwareVersion>
    <SysContact type="String" />
    <SysDescription type="String">Lucent Technologies CBX
    500</SysDescription>
    <SysLocation type="String" />
    <SysName type="String">cbx500f</SysName>
    <SysUpTime type="java.util.Date">Wed Apr 30 16:17:04 IDT 2003</SysUpTime>
    <VendorEnum type="Integer">6</VendorEnum>
  </IManagedElement>
</IManagedElement>
<ID type="Oid">{[ManagedElement(Key=CBX_Sim_Iftach)]}</ID>
<CommunicationStateEnum type="Integer">3</CommunicationStateEnum>
<DeviceName type="String">CBX_Sim_Iftach</DeviceName>
<ElementCategoryEnum type="Integer">2</ElementCategoryEnum>
<ElementTypeEnum type="Integer">22</ElementTypeEnum>
<IP type="com.sheer.types.IPAddress">30.30.30.70</IP>
<InvestigationStateEnum type="Integer">2</InvestigationStateEnum>
<SoftwareVersion type="String">04.02.01.00</SoftwareVersion>
<SysContact type="String" />
<SysDescription type="String">Lucent Technologies CBX
500</SysDescription>
<SysLocation type="String" />
<SysName type="String">cbx500f</SysName>
<SysUpTime type="java.util.Date">Mon May 05 16:16:51 IDT 2003</SysUpTime>
<VendorEnum type="Integer">6</VendorEnum>
</IManagedElement>
</IMObjects_Array>

```

The above XML reply represents a list of two devices (two Lucent CBX switches, with IP address 192.168.2.60 and 30.30.30.70). The data objects in the reply are based on the IMO information model, which is Sheer's internal, generic information model. The above list is an array of IMO objects of type *IManagedElement* (the IMO object that represents a NE).

Each data entity type in the Sheer system has a corresponding IMO object type, which is serialized by BQL into XML strings. The full set of all IMO objects is described in the "IMO Reference Manual". See sections 2 and 3 for further description of IMO.

1.2.3 Device-List Report Example

Following is the full Perl example for retrieving a device list:

```

use IO::Socket::INET;

# ----- Open a BQL session
my $sock = IO::Socket::INET->new(PeerAddr => '192.168.2.110',
    PeerPort => 9002,
    Proto => 'tcp',
    Type => SOCK_STREAM);
die "Cannot open socket" unless defined $sock;

print $sock "openconnection user='John' password='XYZ'\n.\n";
my $Recv;
my $LoginOK=0;
while (($Recv=<$sock>)!~ /^\.\/)
{
    if ($Recv =~ /\.*success.*/)
    {
        print "Login OK\n";
        $LoginOK=1;
    }
}
if (!$LoginOK)
{
    print "Login failed\n";
    print $sock "exit\n.\n";
    $sock->close();
    exit;
}

```

```

}

# ----- Send command and retrieve reply
print $sock "<command name='DeviceList'></command>\n.\n";

my $Result = "";
while (($Recv=<$sock>) !~ /\^\.\/ )
{
    $Result = $Result . $Recv;
}

print $Result, "\n";

print $sock "exit\n.\n";
$sock->close();

```

Note, that the reply to the BQL command is returned from the DNA Gateway socket as a data stream. Therefore, the program has to run a loop of reading from the socket into the temporary variable `$Recv`, and concatenating it into the result variable `$Result`.

1.2.4 The BQL Get Command

The most commonly used BQL command is the **GET** command. It specifies the **OID** (object identifier) of a data entity in the system and returns the information on this object. The OID uniquely identifies any data entity in the system. The GET command can be used not only for reporting a single object, but also for reporting a **Construct** of objects, which is a hierarchy of related data objects (e.g. a NE with all its cards & ports, an end-to-end service path etc.). When retrieving data constructs, the specified OID serves as the “Root” of the construct, from which the related objects are traversed.

Following is an example of a simple GET command:

```

<command name="Get">
  <param name="oid">
    <value>{[ManagedElement(Key=Fore251Lab)]}</value>
  </param>
  <param name="rs">
    <value>
      <key name="DeviceProperties">
        <key name="requiredProperties">
          <key name="*">
            <entry name="*" />
          </key>
        </key>
      </key>
    </value>
  </param>
</command>

```

The query in this example will retrieve the properties of the NE identified by the OID `{[ManagedElement(Key=Fore251Lab)]}`. It has an additional parameter, the **Retrieval Specification** (RS). The RS defines the information scope of the query, by specifying what type of IMO objects to retrieve, what object relationships to traverse, and what properties to include/exclude for each returned object.

The retrieval specification is an XML string with the following format:

```
<param name="rs">
  <value>
    <key name="[rs name]">
      <entry name="register">[true/false]</entry>
      <key name="requiredProperties">
        <key name="[* or IMO type]">
          <entry name="[* or property name]">
            <entry name="[* or property name]">
          </key>
        </key>
      </key>
      <key name="excludedProperties">
        <key name="[* or IMO type]">
          <entry name="[* or property name]">
            <entry name="[* or property name]">
          </key>
        </key>
      </key>
    </key>
  </value>
</param>
```

For example:

```
<param name="rs">
  <value>
    <key name="DeviceProperties">
      <key name="requiredProperties">
        <key name="com.sheer.imo.IManagedElement">
          <entry name="*">
        </key>
      </key>
    </key>
  </value>
</param>
```

In this example the RS specifies the retrieval of only the device properties (IP Address, Type, Vendor, Uptime etc.).

The output of the above BQL GET example (device properties of the NE whose key is “Fore251Lab”) will look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<IManagedElement>
  <ID type="Oid">{[ManagedElement(Key=Fore251Lab)]}</ID>
  <CommunicationStateEnum type="Integer">3</CommunicationStateEnum>
  <DeviceName type="String">Fore251Lab</DeviceName>
  <ElementCategoryEnum type="Integer">2</ElementCategoryEnum>
  <ElementTypeEnum type="Integer">13</ElementTypeEnum>
  <IP type="com.sheer.types.IPAddress">192.168.2.251</IP>
  <InvestigationStateEnum type="Integer">2</InvestigationStateEnum>
  <LogicalRoot type="ILogicalRoot">
    <ID type="Oid">{[ManagedElement(Key=Fore251Lab)][LogicalRoot]}</ID>
  </LogicalRoot>
  <PhysicalRoot type="IPhysicalRoot">
    <ID type="Oid">{[ManagedElement(Key=Fore251Lab)][PhysicalRoot]}</ID>
  </PhysicalRoot>
  <ProvisioningSupportedServices type="java.lang.String_Array">
    <java.lang.String>256_ILink</java.lang.String>
    <java.lang.String>384_Bizlink</java.lang.String>
    <java.lang.String>GSHDSL_256K_cbr</java.lang.String>
  </ProvisioningSupportedServices>
  <SoftwareVersion type="String">S_ForeThought_7.0.0 FCS-Patch
    (1.101625)</SoftwareVersion>
  <SysContact type="String">elil1</SysContact>
  <SysDescription type="String">Marconi ASX-1000</SysDescription>
  <SysLocation type="String">Sheer-labs</SysLocation>
  <SysName type="String">ATM SWITCH</SysName>
  <SysUpTime type="java.util.Date">Mon Apr 14 13:00:50 IDT 2003</SysUpTime>
  <VendorEnum type="Integer">5</VendorEnum>
</IManagedElement>
```

The retrieval specification allows registering for change notifications on the specified objects. When specifying `register=true` in the RS, the DNA platform will monitor changes in all the objects that are defined in the RS scope. Whenever it detects a change in any of the objects it sends an unsolicited notification of the change. Consider, for example, the following notification:

```
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IScalarNotification>
    <ID
      type="Oid">{[Notification(SequenceNumber=1201)(Time=1052837316305)]}</ID>
    <PropertyName type="String">SysName</PropertyName>
    <NewIMO type="IManagedElement">
      <ID type="Oid">{[ManagedElement(Key=ASAMLab)]}</ID>
      <SysName type="String">ATM Switch</SysName>
    </NewIMO>
  </IScalarNotification>
</IMObjects_Array>
```

This notification indicates that the property `SysName` has changed, in the IMO object of type `IManagedElement`, of the NE “ASAMLab”. Further details on Retrieval Specifications and notifications can be found in sections 3.5 and 3.6.2.

Figure 1 summarizes the integration steps using the BQL. The steps are:

1. Open TCP socket to the BQL interface
2. Authenticate using username and password
3. Send BQL commands to the Sheer DNA Gateway
4. Receive & parse the returned results
5. Optional: Register and receive change notifications
6. Close the connection with the BQL interface.

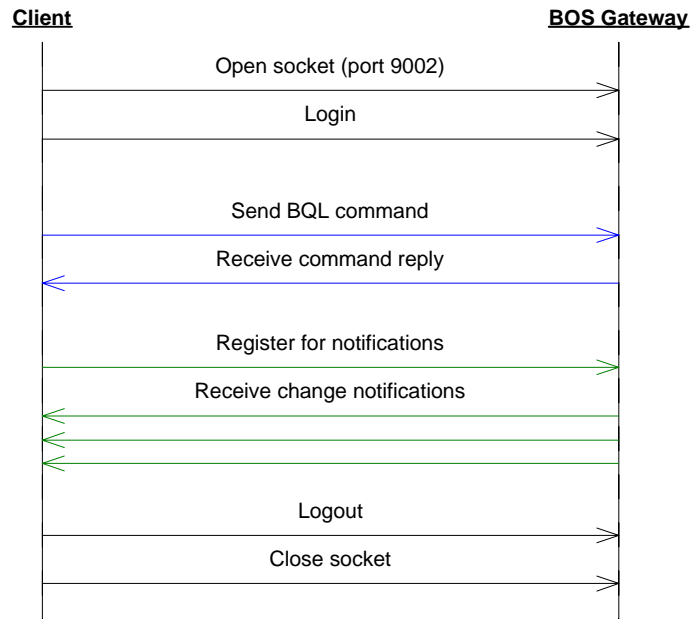


Figure 1: Integration Sequence

2 Understanding IMO

2.1 What is IMO?

The *Information Model Objects* (IMO) framework in Sheer DNA is a standard for generic information representation. It defines representation of multi-Vendor, multi-technology, multi-layer network & service information, based on the TMF-513/608 MTNM (Multi-Technology Network Management) recommendations. Internally within DNA IMO is implemented as a set of interface classes for representing all network & service information objects. However, for external integration purposes, the IMO specification defines 1:1 bi-directional translation of all IMO objects to XML. External applications that access Sheer DNA via BQL, are exposed only to the XML formatting of the DNA information.

It is important to note, that IMO is only a temporary “Packaging” mechanism. The live network data model maintained in Sheer DNA is stored within the Virtual Network Element (VNEs) in the form of DC (*Device Component*) hierarchies, whereas persistent information (e.g. event/alarm history, Service identifiers etc.) is kept in the DNA database. When interfacing with external systems, DNA works with transient IMO objects, which translate to/from the actual data elements in the system. This enables sending information from/to DNA in a generic, consistent way (analogous to SNMP, in which the MIB is not the actual information repository, but only a data reference map, interacting with client applications by through transient PDUs).

2.2 IMO Basics

IMO is the data schema of the network. All network & service information is modeled in IMO objects. Each IMO object has a unique *Object Identifier* (OID), analogous to OIDs in SNMP MIBs.

Each IMO object has *Properties*, which contain the actual data. The properties can be of several types:

- Primitive types (e.g. string, integer, etc.)
- Sheer basic types (e.g. IP address, MAC address etc.)
- References to other nested IMO objects.

The properties can appear as single scalars (e.g. IP address of an NE) or as part of an array (e.g. list of Physical Ports in a Card).

IMO objects may also contain *Aspects*, which are dynamic properties used to extend IMO objects at runtime. This mechanism enables extending IMO objects (representing network resources) with additional information from the DNA database. For example, an IMO object representing a port can be extended with an Aspect that contains subscriber information. Another example is an IMO Construct of a NE with all its associated alarms: the IMO object representing the NE is extended with Aspects, which contain the associated alarms.

As described above, most IMO objects contain references to other, related IMO objects. These relations describe network dependencies, such as:

- **Containment** – e.g. Cards within a NE, Ports within a Card
- **Logical reference** – e.g. Traffic profile associated to a circuit
- **Connectivity** – e.g. physical connection between ports, cross-connection between circuits

The inter-object references enable DNA to pack and provide IMO data in the form of object *Constructs*. A Construct is a set (hierarchy) of inter-related objects. Every IMO Construct has a “root” object, which is the entry point to the objects collection, through which the construct can be traversed (“surfing”).

Note, that the object Constructs can contain any graph of objects (supporting references in any direction), and not just “top-down” hierarchical trees. For example, an IMO Construct whose root object is a specific NE Card, may contain a reference to the Chassis in which it resides and which is its ancestor in the physical network hierarchy.

2.3 De-referencing

Each IMO has a unique id call IOid. When one IMO is serving as a property of another IMO, sometimes there is no direct Object and property relation between them but a reference relation using the IOid. For example, instead of the IMO instance serving as a property of another, there is an IOid instance representing the IMO instance. In this case, in order to retrieve the data, two queries are required, for example:

When we execute a "Get" command on IVirtualRouter, observing its "get" methods we see it contains a method called "getVrf()". It would be expected that when invoked this will retrieve an IVrf, but actually this method return type returns IVrfOid, e.g. the id of the IVrf. To retrieve the IVrf data we need to execute another "get" command using the IVrfOid to retrieve the IVrf data. "De-Referencing" is a method of retrieving the IVirtualRouter data including the referenced IVrf in one query, retrieving the IVrf data as an aspect of IVirtualRouter.

In the required aspects of the RetrievalSpecification, insert the "com.sheer.imo.keys.IReferencedImosAspectOid" in the IMO to be de-referenced:

```
<key name="req_aspects">
.
.
.
  <key name="com.sheer.imo.keys.IVirtualRouterOid">
    <entry name="com.sheer.imo.keys.IReferencedImosAspectOid"></entry>
  </key>
</key>
```

Specify the de-referenced method inside the required properties, as follows:

```
<key name="req_prop">
.
.
.
  <key name="com.sheer.imo.IReferencedImosAspect">
    <entry name="Vrf"></entry>
  </key>
</key>
```

Specify the properties to be retrieved on the de-referenced IMO, in this example, the properties to be retrieved on the IVrf:

```
<key name="req_prop">
.
.
.
  <key name="com.sheer.imo.IReferencedImosAspect">
    <entry name="Vrf"></entry>
  </key>
  <key name="com.sheer.imo.IVrf">
    <entry name="*"></entry>
  </key>
</key>
```

Thus, one query is performed, and the IVrf is received as an aspect on the IVirtualRouter.

2.4 IMO by Example

This section provides some examples of IMO objects & constructs. It describes their contents structure as well as their XML encoding.

2.4.1 Simple IMO object Example

Consider the following IMO object:

```
<technologies.IAdsl>
  <ID type="Oid">{[ManagedElement(Key=DallasPE)][PhysicalRoot]
    [Chassis][Shelf(ShelfNum=1)][Slot(SlotNum=LT3)][Module]
    [Port(PortNumber=9)][PhysicalLayer]}
  </ID>
  <AdminStatusEnum type="Integer">1</AdminStatusEnum>
  <OperStatusEnum type="Integer">2</OperStatusEnum>
  <MaxSpeed type="com.sheer.types.Speed">0.0 bps</MaxSpeed>
  <TypeEnum type="Integer">94</TypeEnum>
</technologies.IAdsl>
```

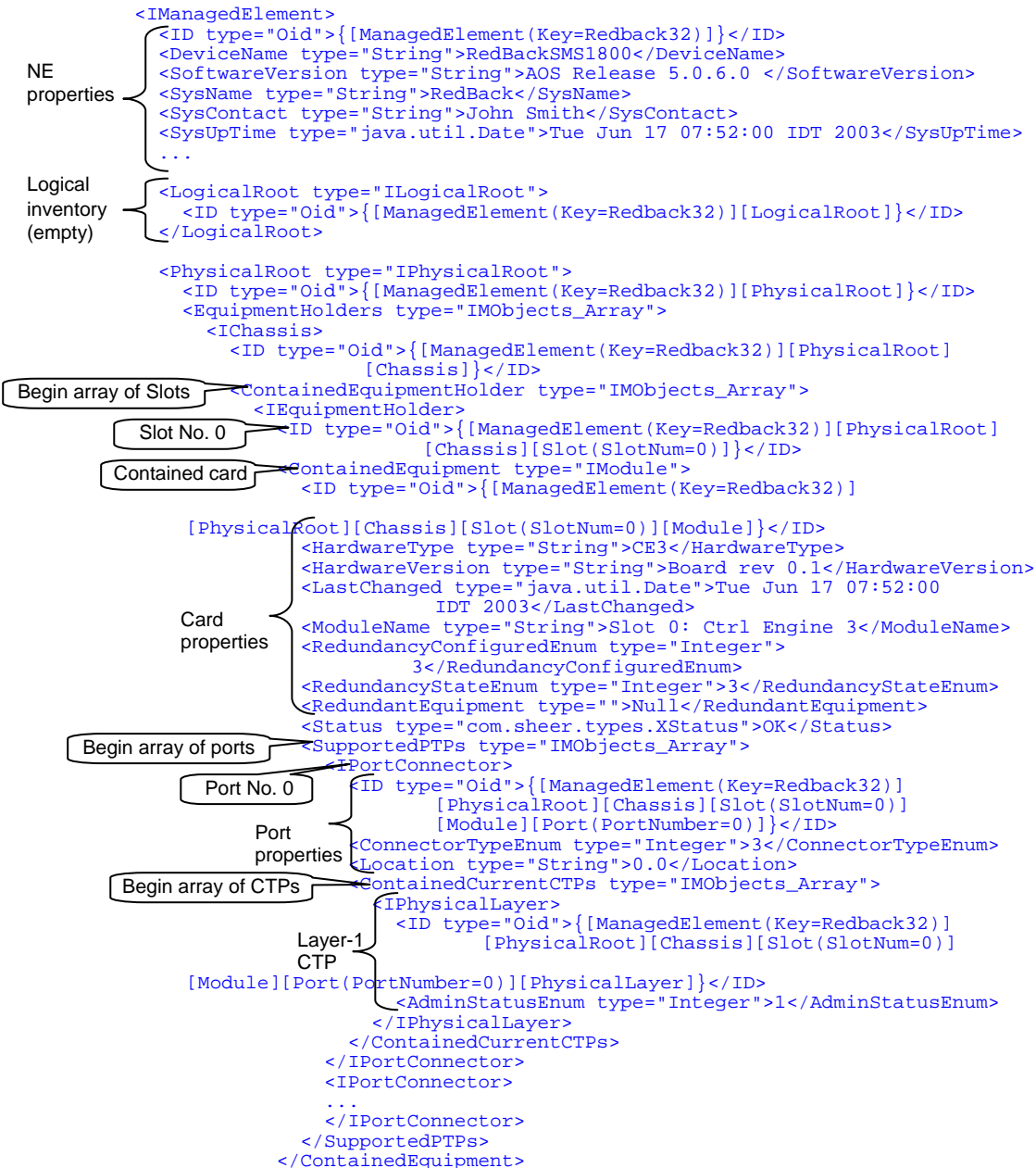
The IMO object type is `technologies.IAdsl` and represents the layer-1 properties of an ADSL port. The following table summarizes the properties contained in this IMO object:

Property	Type	Value	Meaning
ID	Oid	{ManagedElement(Key=DallasPE)][PhysicalRoot][Chassis][Shelf(ShelfNum=1)][Slot(SlotNum=LT3)][Module][Port(PortNumber=9)][PhysicalLayer]}	Layer-1 properties of port 9 in slot LT3 in shelf 1 in the NE "DallasPE"
AdminStatusEnum	Integer	1	Up
OperStatusEnum	Integer	2	Down
MaxSpeed	com.sheer.types.Speed	0.0 bps	A Sheer basic type, which contain the speed (in this case, 0.0) and the units (bps)
TypeEnum	Integer	94	IANA protocol type

The full details and enumerations of every IMO object type can be looked up in the IMO reference document.

2.4.2 IMO Construct Example

Following is an example of a physical Inventory IMO Construct, specifying a NE with its contained Cards, Ports & CTPs:



```

Next Slots {
  </IEquipmentHolder>
  <IEquipmentHolder>
    <ID type="Oid">{ [ManagedElement (Key=Redback32)] [PhysicalRoot]
      [Chassis] [Slot (SlotNum=2)] }</ID>
    ...
  </IEquipmentHolder>
  <IEquipmentHolder>
    <ID type="Oid">{ [ManagedElement (Key=Redback32)] [PhysicalRoot]
      [Chassis] [Slot (SlotNum=3)] }</ID>
    ...
  </IEquipmentHolder>
  <IEquipmentHolder>
    <ID type="Oid">{ [ManagedElement (Key=Redback32)] [PhysicalRoot]
      [Chassis] [Slot (SlotNum=4)] }</ID>
    ...
  </IEquipmentHolder>
  </ContainedEquipmentHolder>
  </IChassis>
  </EquipmentHolders>
  </PhysicalRoot>
  </IManagedElement>
}

```

In this example, the Construct provides a complete hierarchy of the device physical inventory. Each level (NE, Card, Port etc.) contains its own properties, as well as an array of nested objects (if relevant).

The root IMO object of the Construct is the NE itself, which is of IMO type `IManagedElement`. It contains the list of NE properties and an array of Chassis elements, which are of type `IChassis` (in this device, there is only one Chassis). The Chassis object contains an array of slots (of type `IEquipmentHolder`). Each slot has a single property (`ContainedEquipment`), which is an IMO object representing a Card (IMO type `IModule`). Each card contains an array of ports (`IPortConnector`) which contain the CTPs (Connection Termination Points – represent the specific layers in the port's communication stack). In this case, there is only one CTP (layer-1 represented by `IPhysicalLayer`).

Note, that the NE object (`IManagedElement`) contains an IMO reference for the logical inventory branch (`ILogicalRoot`), which in this example is empty. This branch can contain all the logical inventory of the device. Here it is empty since it was filtered out in the Retrieval Specification.

2.5 Typical IMO Constructs

2.5.1 Inventory Containment

A typical NE physical inventory is modeled in IMO as follows:

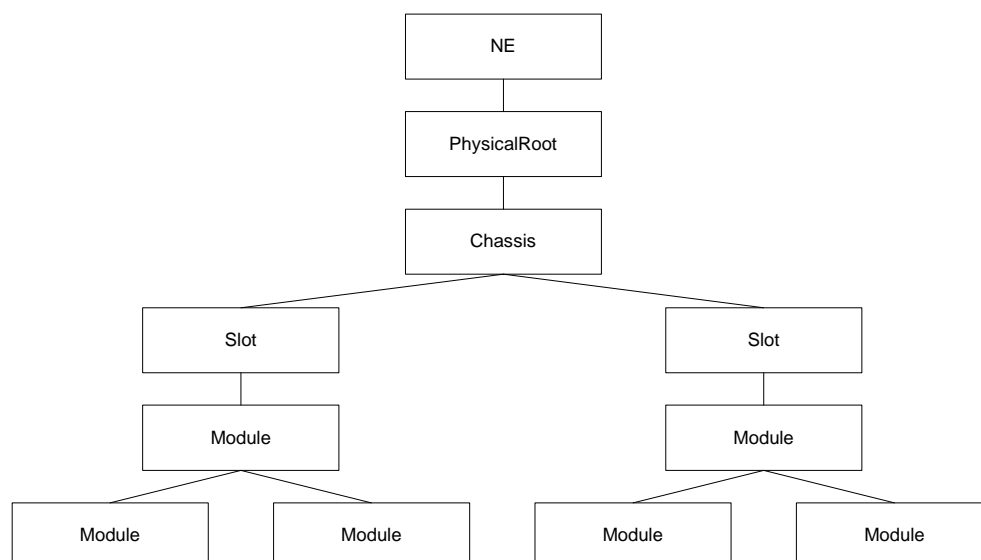


Figure 2: IMO Modeling for Inventory Containment

Note, that different NE types may have a different IMO inventory hierarchy, i.e. a deeper hierarchy of shelves, modules & sub-modules.

2.5.2 Port Connections

IMO models a port as a construct. The root object is the physical port (PTP – Physical Termination Point), which is of type `IPortConnector`. It contains a hierarchy of CTPs (Connection Termination Points) which are IMO objects representing the connections at the specific network layers. The CTP hierarchy is mapped to the network stack - every CTP at layer N contains all its associated CTPs at layer N+1. In addition, CTPs can also contain related logical entities (e.g., traffic profiles). This type of modeling follows the TMF-513/608 recommendations.

This relationship (CTP→Traffic profile) demonstrates how in IMO the cross relationship between objects is based on the network functionality and not restricted to “classic” hierarchy. Objects have multiple relations with other objects in all directions, and can be constructed and contained within each other in various ways.

See Figure 3 for some examples of port modeling.

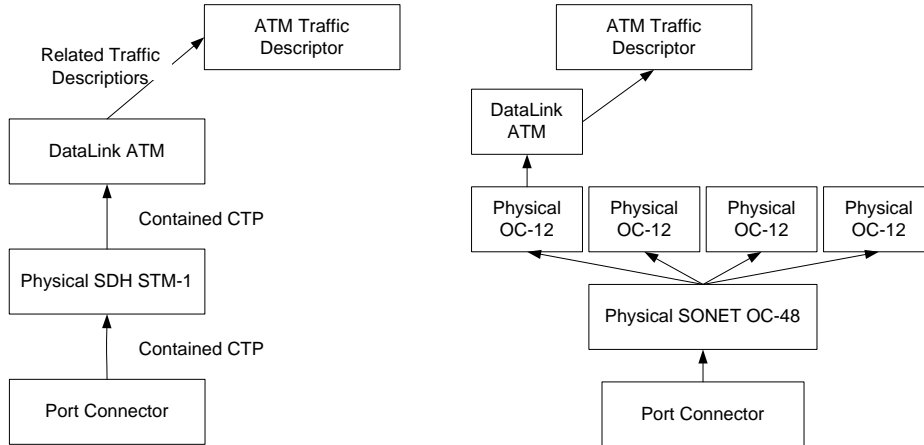


Figure 3: IMO Port Modeling

Following is an example with the XML representation of the modeling in Figure 3:

```

<IPortConnector>
  <Location type="String">1.NTA.1</Location>
  <ContainedCurrentCTPs type="IMObjects_Array">
    <technologies.ISonetSdh>
      <AdminStatusEnum type="Integer">1</AdminStatusEnum>
      <ContainedCurrentCTPs type="IMObjects_Array">
        <technologies.IAtm>
          <Bandwidth type="technologies.IAtmBandwidthTrafficDescriptor">
            <TxMaxBandwidth type="com.sheer.types.Speed">
              0.0 bps</TxMaxBandwidth>
            <TxUbrAllocBandwidth type="com.sheer.types.Speed">
              13.86 Mbps</TxUbrAllocBandwidth>
          </Bandwidth>
          <VcsTableSize type="Integer">50</VcsTableSize>
          <VpVcRange type="String">[VP: 0..4096, VC: 0..65536]</VpVcRange>
        </technologies.IAtm>
      </ContainedCurrentCTPs>
    </technologies.ISonetSdh>
  </ContainedCurrentCTPs>
</IPortConnector>
  
```

2.5.3 Service Paths (SNCs)

IMO models Sub-Network Connections (SNCs) according to the recommendations TMF-513. The general object hierarchy that represents a SNC is described in Figure 4.

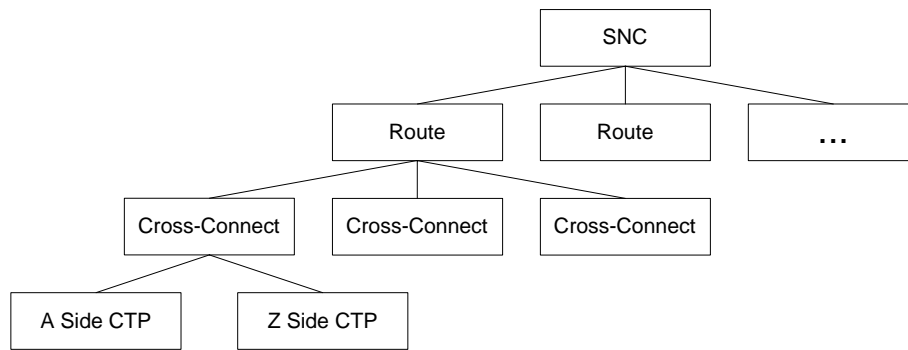


Figure 4: IMO Modeling for SNC

The root of the hierarchy is the SNC object, representing the entire service path. The SNC contains multiple “Route” objects, one for each networking layer (defining the end-to-end path for that layer). Each Route object contains a list of Cross-Connect objects, representing the cross-connection in each NE along the path (according to the network layer), and each cross-connect contains its respective A side and Z side CTP objects.

Note: As before, the objects in the SNC hierarchy contain references that can be traversed to pull in additional related data objects, such as traffic descriptors or port parameters.

Following is an example of a DSL/ATM service path in Figure 5:

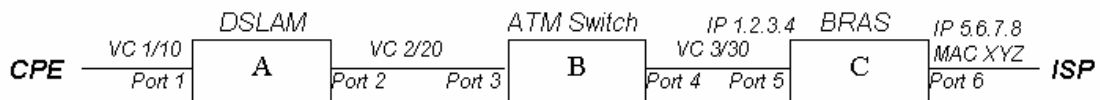


Figure 5: DSL/ATM Service Path

In this example, the DSL service starts from an ATM VC on the DSL port and terminates on an IP interface in the BRAS. The relevant cross-connections are in Layer 2 (up to the BRAS) and Layer 3 (in the BRAS), as described below:

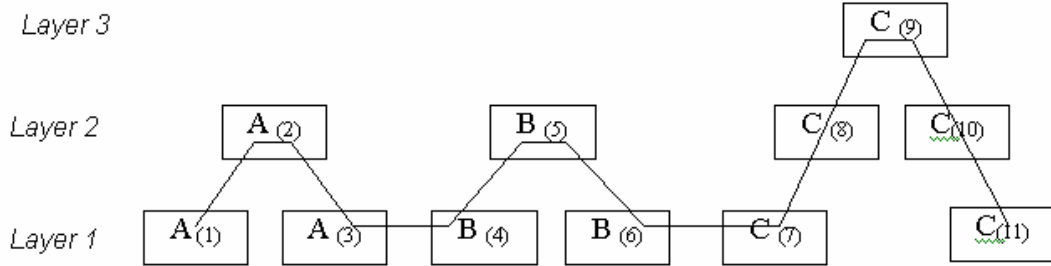


Figure 6: Cross Connect Sequence

The IMO representation of this SNC is given in Figure 7 below:

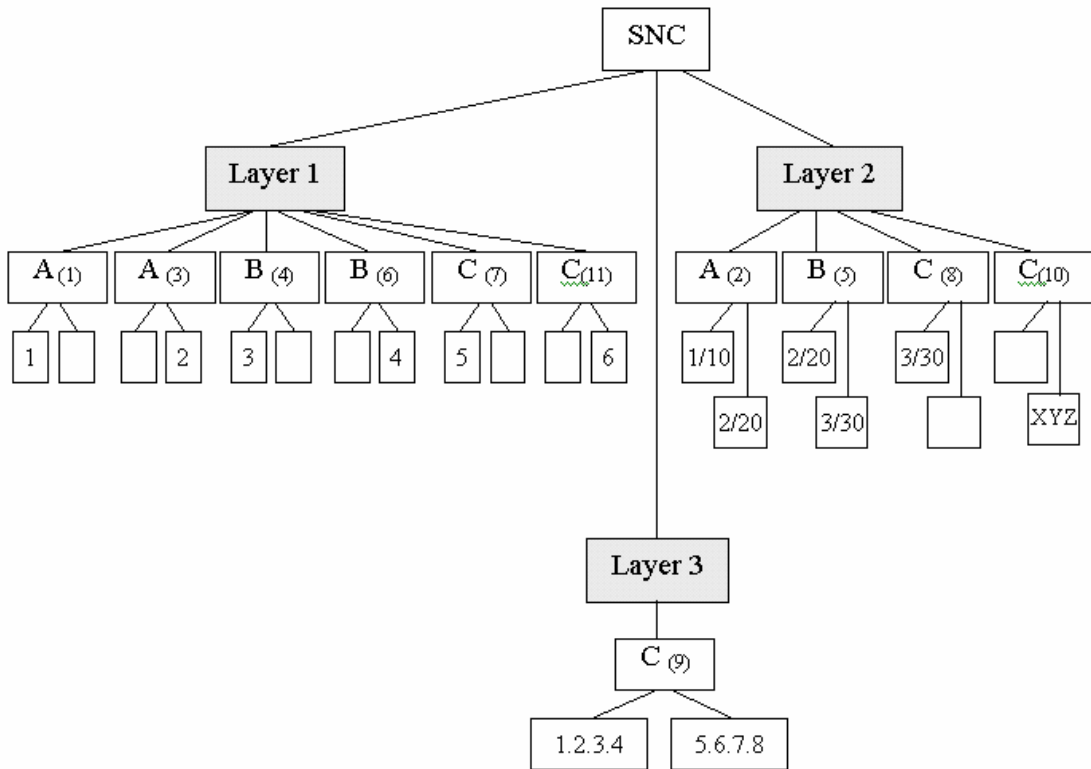


Figure 7: IMO Representation for SNC

The path is defined as an ordered list of cross-connects. Each Cross-Connect object contains a running index (shown above in parentheses inside the Cross-connect blocks), starting at 1. The indices determine the order of cross-connects constituting the path (in this example from 1 to 11).

Note, that for tracing an end-to-end path it would have been enough to model only the cross-connects of the layer in which the service path is routed in each device.

This would mean, that in the above example it would have been enough to provide only Layer-2 cross-connects for the DSLAM & ATM Switch (devices A & B) and Layer-3 cross-connects for the BRAS (device C). Still, The SNC IMO provides the information of all layers, in order to provide richer information and a fuller context for the SNC.

3 IMO Specification

3.1 IMO Format/Syntax

The general XML format of an IMO object is as follows:

```

IMO:                <IMO_TYPE>
                    IMO_OID
                    IMO_PROPERTIES
                    </IMO_TYPE>

IMO_OID:            <ID type="Oid"> OID_FORMAT </ID>
IMO_PROPERTIES:    SIMPLE_PROPERTY | PROPERTIES_ARRAY
SIMPLE_PROPERTY:   <PROPERTY_NAME type=PROPERTY_TYPE>
                    PROPERTY_VALUE
                    </PROPERTY_NAME>
PROPERTIES_ARRAY: <PROPERTY_NAME type=IMObjects_Array>
                    IMO
                    < IMObjects_Array>

OID_FORMAT:        A valid string encoding of an OID
PROPERTY_NAME:    String representing property name
PROPERTY_TYPE:    String representing a valid type.
                  This can be either a primitive type
                  or an IMO type.
PROPERTY_VALUE:   String representation of a property
                  value

```

3.2 IMO OIDs

An IMO OID (Object Identifier) is the unique identifier of every IMO instance in the system. The OID uniquely identifies every IMO object, by providing a cascaded structure that describes the location of the entity. For example, the OID of a specific port in a typical NE will be formatted by cascading the NE IP address, Shelf number, Module number and Port number.

Different device types may have different OID schemas. For example, a port OID in some NE may also include Sub-Modules, Sub-Slots etc.

3.2.1 Format/Syntax

```
OID:                {INTERFACE_LIST}
INTERFACE_LIST:    INTERFACE | INTERFACE INTERFACE_LIST
INTERFACE:         [INTERFACE_NAME] |
                  [INTERFACE_NAME(INTERFACE_PARAMS)]
INTERFACE_NAME:    String representation of a physical
                  hierarchy in the device
INTERFACE_PARAMS:  KEY=VALUE |
                  KEY=VALUE, INTERFACE_PARAMS
KEY                String
VALUE              String
```

3.2.2 OID Example

The OID of Port 3 on module 1, sub-module C of the NE “London5” will be:

```
{ [ManagedElement (Key=London5)] [PhysicalRoot] [Chassis]
  [Slot (SlotNum=1)] [Module] [Slot (SlotNum=C)] [Module]
  [Port (PortNumber=3)] }
```

The hierarchy of this OID is illustrated in Figure 8.

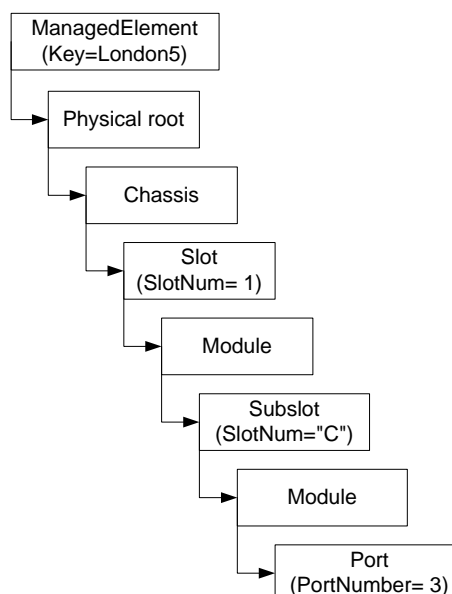


Figure 8: OID Hierarchy

3.2.3 Sample OID Hierarchies

As described above, each NE type may have its own IMO schema, i.e. different object types and tree hierarchy. In order to be able to specify the correct OID for each NE, the client application has to be familiar with the OID hierarchy of the relevant NE. Following are some examples of IMO hierarchies.

3.3.2.1 Alcatel ASAM – Physical Object

The OID of a port of an Alcatel ASAM is encoded as

```
{ [ManagedElement (Key=ASAMLab)] [PhysicalRoot] [Chassis]
  [Shelf (ShelfNum=1)] [Slot (SlotNum=NTA)] [Module]
  [Port (PortNumber=1)] }
```

The physical hierarchy represented by this OID is illustrated in Figure 9.

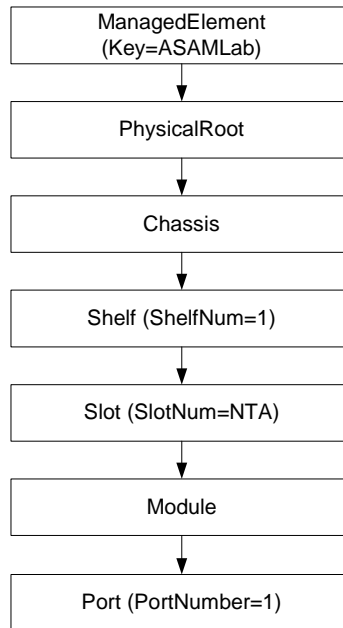


Figure 9: Alcatel ASAM Port OID

3.3.2.2 Lucent CBX – Physical Object

The OID of a port of a Lucent CBX is encoded as

```

{ [ManagedElement (Key=CBX500Lab)] [PhysicalRoot] [Chassis]
  [Slot (SlotNum=12)] [Module] [Port (PortNumber=6)] }
  
```

The physical hierarchy represented by this OID is illustrated in Figure 10.

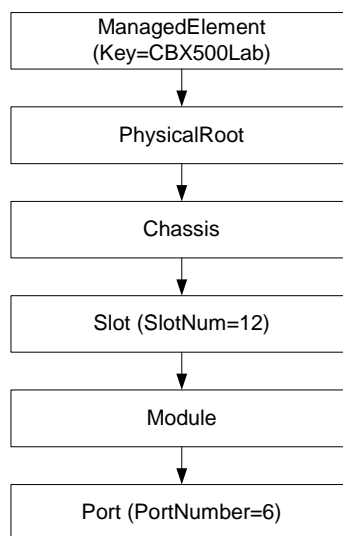


Figure 10: Lucent CBX Port OID

3.3.2.3 Redback SMS – Physical Object

The OID of a port of a Redback SMS is encoded as:

```
{ [ManagedElement (Key=RB1)] [PhysicalRoot] [Chassis]  
  [Slot (SlotNum=7)] [Module] [Port (PortNumber=1)] }
```

The physical hierarchy represented by this OID is illustrated in Figure 11.

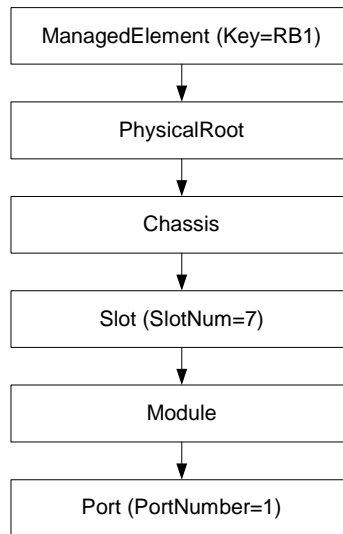


Figure 11: RedBack SMS Port OID

3.3.2.4 Redback SMS – Logical Object

The OID of a logical object (IP interface) of a RedBack SMS is encoded as

```
{ [ManagedElement (Key=RB1)] [LogicalRoot]
  [Context (ContextName=cyberworks.net.sg)] [RoutingEntity]
  [IpInterface (IPAddress=10.254.253.37)] }
```

The logical hierarchy represented by this OID is illustrated in Figure 12:

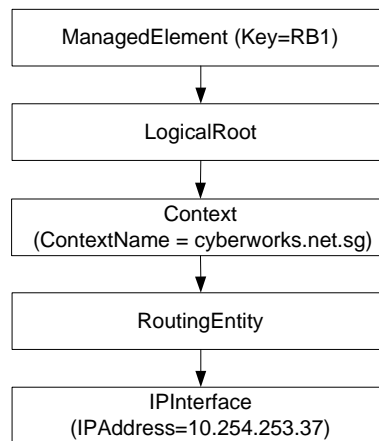


Figure 12: RedBack SMS Logical OID

3.3 IMO Primitives

When encoding an IMO object, each of its properties may be either a primitive (i.e. basic type) or another IMO. The simplest case is when all the properties of the IMO are primitives, as in the following example:

```
<IManagedElement>
  <ID type="Oid">{[ManagedElement (Key=CBX500Lab)]}</ID>
  <CommunicationStateEnum type="Integer">3
    </CommunicationStateEnum>
  <DeviceName type="String">CBX500Lab</DeviceName>
  <ElementCategoryEnum type="Integer">2
    </ElementCategoryEnum>
  <ElementTypeEnum type="Integer">22</ElementTypeEnum>
  <IP type="com.sheer.types.IPAddress">192.168.2.70</IP>
  <InvestigationStateEnum type="Integer">2
    </InvestigationStateEnum>
  <SoftwareVersion type="String">04.02.01.00
    </SoftwareVersion>
  <VendorEnum type="Integer">6</VendorEnum>
</IManagedElement>
```

The following table describes the primitive types that can be used for IMO object properties:

Type	Description	Example
Int	4 Bytes integer	1
Long	8 Bytes integer	100
String		“John Smith”
Counter	Counter representation	Octet counter: <long> octets Packet counter: <long> packets
Ippaddress	IP address	192.168.2.23
Macaddress	MAC address	ABAECD FCBBCD
Ipsubnet	IP address, mask ip address	202.79.94.0, 255.255.255.192
Key	String identifier	“Redback17”
Speed	Speed representation with units	<long> Cells/sec <long> bps <long> Kbps <long> Mbps <long> Gbps <long> Cell/sec
Rate	Rate representation with units	<long> bps <long> Kbps <long> Mbps <long> Gbps <long> Bps <long> KBps <long> MBps <long> GBps <long> Cell/sec
vpvcrange	Define available range for VCs	If minimal vp=maximal vp: VP: <integer min vp>, VC: <integer min vc>..<integer max vc>

Type	Description	Example
		Otherwise: VP: <integer min vp>.. VP: <integer max vp>, VC: <integer min vc>.. VC: <integer max vc>
vprange	Available vps range	<integer min vp>..<integer max vp>
vpvcrangecollection	A collection of available VCs ranges	[vpvcrange][vpvcrange]...

3.4 IMO Constructs

IMO constructs support recursive nesting of IMO objects as well as arrays of IMO objects. First, we will describe how to how to nest an IMO object within another IMO object. We will then describe how to aggregate several IMO objects into an IMO objects array, and finally, we will describe how to combine both to create complex IMO constructs.

3.4.1 Nesting IMO Objects

Each of the properties of an IMO object can be an IMO object by itself. In this case the type of the property is one of the existing IMO types. For example:

```
<technologies.IAtm>
  <Bandwidth type="technologies.IAtmBandwidthTrafficDescriptor">
    <TxMaxBandwidth type="com.sheer.types.Speed">0.0 bps
    </TxMaxBandwidth>
    <TxUbrAllocBandwidth type="com.sheer.types.Speed">13.86 Mbps
    </TxUbrAllocBandwidth>
  </Bandwidth>
</technologies.IAtm>
```

In this example, the IMO object of type `technologies.IAtm` contains a property called `Bandwidth`, which is an IMO object of the type `technologies.IAtmBandwidthTrafficDescriptor`.

Following is another example, which combines both primitives and nested IMOs:

```
<technologies.IAtm>
  <Bandwidth type="technologies.IAtmBandwidthTrafficDescriptor">
    <TxMaxBandwidth type="com.sheer.types.Speed">0.0 bps
    </TxMaxBandwidth>
    <TxUbrAllocBandwidth type="com.sheer.types.Speed">13.86 Mbps
    </TxUbrAllocBandwidth>
  </Bandwidth>
  <VcsTableSize type="Integer">50</VcsTableSize>
  <VpVcRange type="String">[VP: 0..4096, VC: 0..65536]</VpVcRange>
</technologies.IAtm>
```

3.4.2 IMO Object Arrays

The general format of the encoding of an IMO array is

```
<IMObjects_Array>
  IMO1
  IMO2
  ...
</IMObjects_Array>
```

Following is an example of an IMO construct, which includes an array of two IMO objects of the type `IManagedElement`:

```
<IMObjects_Array>
  <IManagedElement>
    <ID type="Oid">{[ManagedElement(Key=CBX500Lab)]}</ID>
    <DeviceName type="String">CBX500Lab</DeviceName>
    <CommunicationStateEnum type="Integer">3
      </CommunicationStateEnum>
    </IManagedElement>
  <IManagedElement>
    <ID type="Oid">{[ManagedElement(Key=CBX_Iftach)]}</ID>
    <CommunicationStateEnum type="Integer">3
      </CommunicationStateEnum>
    <DeviceName type="String">CBX_Iftach</DeviceName>
    </IManagedElement>
</IMObjects_Array>
```

3.4.3 Complex Constructs

In the next example, we combine the properties of type IMO with IMO arrays to create a nested construct of IMO objects:

```
<IVrf>
  <ID type="Oid">{[ManagedElement(Key=DallsPE)][LogicalRoot]
    [FWComponentContainer(Type=3)][Vrf(VrfName=ACME)]}</ID>
  <Name type="String">ACME</Name>
  <RouteDistinguisher type="IRouteDistinguisher">
    <ID type="Oid">{[ManagedElement(Key=DallsPE)][LogicalRoot]
      [FWComponentContainer(Type=3)][Vrf(VrfName=ACME)]
      [RouteDistinguisher]}</ID>
    <RouteDistinguisher type="String">3603:1</RouteDistinguisher>
  </RouteDistinguisher>
  <VrfTable type="IMObjects_Array">
    <technologies.IVrf>
      <ID type="Oid">{[ManagedElement(Key=DallsPE)][LogicalRoot]
        [FWComponentContainer(Type=3)][Vrf(VrfName=ACME)]
        [VrfEntry(IP=172.16.101.0)]}</ID>
      <BGPNextHop type="">Null</BGPNextHop>
      <InnerLabel type="">Null</InnerLabel>
      <NextHop type="IPAdderss">172.16.101.254</NextHop>
      <RouteDest type="IPAdderss">172.16.101.0</RouteDest>
      <RouteMask type="IPAdderss">255.255.255.0</RouteMask>
      <RouteProtocolTypeEnum type="Integer">14
        </RouteProtocolTypeEnum>
      <TypeEnum type="Integer">3</TypeEnum>
    </technologies.IVrf>
  </VrfTable>
  <technologies.IVrf>
    <ID type="Oid">{[ManagedElement(Key=DallsPE)][LogicalRoot]
      [FWComponentContainer(Type=3)][Vrf(VrfName=ACME)]
      [VrfEntry(IP=192.168.107.207)]}</ID>
    <BGPNextHop type="">Null</BGPNextHop>
    <InnerLabel type="">Null</InnerLabel>
    <NextHop type="">Null</NextHop>
    <RouteDest type="IPAdderss">192.168.107.207</RouteDest>
    <RouteMask type="IPAdderss">255.255.255.255</RouteMask>
    <RouteProtocolTypeEnum type="Integer">14
      </RouteProtocolTypeEnum>
    <TypeEnum type="Integer">3</TypeEnum>
  </technologies.IVrf>
</VrfTable>
</Vrf>
```

3.5 Retrieval Specifications

The Retrieval Specification (RS) defines the scope of information to be retrieved by a GET or FIND command. It describes the IMO objects that will be returned as well as the properties that will be returned in each IMO object. The retrieval specification allows us to specify the inclusion or exclusion of every element in the result.

The RS also allows registering for change notifications to be sent whenever a property changes within the specified data scope. Registration for changes enables, for example, receiving notifications on port or module status change, or any changes in the configuration of the network.

The retrieval spec has another section that refers to *Aspects*. We will describe Aspects in section 3.6.

3.5.1 Defining the Retrieval Scope

The retrieval specification is a XML string that represents the scope and structure of the result of a GET query. It defines what IMO objects should be included in the answer and what properties should be included for each IMO object type. Consider, for example, the following simple retrieval spec:

```
<param name="oid">
  <value>{[ManagedElement(Key=DallasPE)]}</value>
</param>
<param name="rs">
  <value>
    <key name="RS1">
      <key name="requiredProperties">
        <key name="com.sheer.imo.IManagedElement">
          <entry name="*" />
        </key>
      </key>
    </key>
  </value>
</param>
```

The above retrieval spec defines that:

- The answer will contain only objects of type `IManagedElement`
- For each object of that type, it will include all its properties.

The name of the retrieval spec (“RS1”) is an optional string identifier that is used only for readability of the XML and ignored when the query is processed.

The following example demonstrates the general format of retrieval specification. It lists the IMO types and the properties within each IMO type that will be included in the reply:

```
<param name="rs">
  <value>
    <key name="RS_name">
      <key name="requiredProperties">
        <key name=IMO_TYPE_A>
          <entry name=PROPERTY_1/>
          <entry name=PROPERTY_2/>
          ...
        </key>
        <key name=IMO_TYPE_B>
          <entry name=PROPERTY_1/>
          <entry name=PROPERTY_2/>
          ...
        </key>
      </key>
    </value>
  </param>
```

To include all IMO types in the result use "*" as the IMO type. Similarly, to include all properties of a given IMO type use "*" as the property name:

```
<param name="rs">
  <value>
    <key name="RS1">
      <key name="requiredProperties">
        <key name="* ">
          <entry name="*" />
        </key>
      </key>
    </value>
  </param>
```

In this example, we specified the "*" wildcard twice. First to indicate that we are interested in all IMO types, and then to indicate that we want all properties in the objects. If we were specifying only the IMO type wildcard, then we would have received empty IMO objects with no properties.

3.5.1.1 Specifying the Retrieved IMO Types

BQL determines what IMO types to retrieve, by recursively traversing the containment (nesting) references between the IMO objects in the DNA information model, starting from the root object OID (specified as part of the GET command). During the recursive traverse, each object that is found is tested against the IMO types in the RS, and is retrieved only if it matches any of these types. Consider again the example:

```
<param name="rs">
<value>
<key name="RS_name">
  <key name="requiredProperties">
    <key name="IMO_TYPE_A">
      <entry name="PROPERTY_1"/>
      <entry name="PROPERTY_2"/>
      ...
    </key>
  </key>
  <key name="IMO_TYPE_B">
    <entry name="PROPERTY_1"/>
    <entry name="PROPERTY_2"/>
  </key>
</key>
</value>
</param>
```

Assume that IMO_TYPE_A contains IMO_TYPE_C as one of its properties. Since we requested in the answer all properties of IMO_TYPE_A, then, IMO_TYPE_C will appear in the result (since it is a property of IMO_TYPE_A). However, it will appear empty (i.e., without properties, except for its OID), since IMO_TYPE_C is not included in the retrieval spec.

To demonstrate this, look at the following RS example:

```
<param name="oid">
  <value>{ [ManagedElement(Key=DallasPE)] }</value>
</param>
<param name="rs">
  <value>
    <key name="RS1">
      <key name="requiredProperties">
        <key name="com.sheer.imo.IManagedElement">
          <entry name="*" />
        </key>
      </key>
    </key>
  </value>
</param>
```

In this example, the RS specifies the inclusion of the IMO type `IManagedElement`, with all its properties. Since the root object (the NE) is of type `IManagedElement`, the NE object is returned with all its properties. However, some of the NE properties are by themselves IMO objects, e.g. `PhysicalRoot` and `LogicalRoot` (of the IMO types `IPhysicalRoot` and `ILogicalRoot`, respectively). Since the RS does not specify these IMO types in the “includedProperties” section, they will appear empty (i.e. appear as properties of the `IManagedElement` object, with their OID only):

```

<IManagedElement>
  <ID type="Oid">{[ManagedElement(Key=Redback32)]}</ID>
  <DeviceName type="String">RedBackSMS1800</DeviceName>
  <SoftwareVersion type="String">AOS Release 5.0.6.0 </SoftwareVersion>
  <SysName type="String">RedBack</SysName>
  <SysContact type="String">John Smith</SysContact>
  <SysUpTime type="java.util.Date">Tue Jun 17 07:52:00 IDT 2003</SysUpTime>
  ...
  <LogicalRoot type="ILogicalRoot">
    <ID type="Oid">{[ManagedElement(Key=Redback32)][LogicalRoot]}</ID>
  </LogicalRoot>
  ...

```

NE properties {

Logical inventory (empty) {

However, had we included `IPhysicalRoot` or `ILogicalRoot` in the required IMO types, BQL would have traversed these objects as well, retrieve their properties (and then continue to test their contained objects against the included IMO types, and so on recursively).

The above examples demonstrate the recursive nature of the retrieval spec. BQL browses through all the nested IMO objects in the DNA information model, and tests them against the specified IMO types in the RS.

An important factor of the recursive nature of the retrieval spec is to have a stopping criterion for information retrieval. When retrieving an IMO object, the system recursively traverses all properties of the retrieved IMO, and for each property, which is a nested IMO object it continues to traverse it. Hence, the included IMO types in the RS should be designed in a way that controls the depth of the returned data.

3.5.1.2 IMO Type Inheritance

IMO objects are defined through an inheritance scheme. For example, CTP objects, such as `IAtm`, `IAdsl` etc. are all derived from (children of) a common base type, `IConnectionTerminationPoint`. It is enough to specify a base type in the RS to have all the derived types implicitly included.

Consider the following example. Assume we have two IMO types (IMO_TYPE_A and IMO_TYPE_B), where IMO_TYPE_B inherits from (is the child of) IMO_TYPE_A. Now assume we execute a GET command in which the root object is of type IMO_TYPE_B, with the following retrieval spec:

```
<param name="oid">
  <value>{Some-object-of-type-B}}</value>
</param>
<param name="rs">
<value>
<key name="RS3">
  <key name="requiredProperties">
    <key name=IMO_TYPE_A>
      <entry name=PROPERTY_1/>
      <entry name=PROPERTY_2/>
    </key>
  </key>
</key>
</value>
</param>
```

When processing the properties of the root object, the system inspects the retrieval spec and detects that its IMO type (IMO_TYPE_B) is not included. However, it detects that its ancestor (IMO_TYPE_A) is included in the retrieval spec, and the object is qualified for retrieval (with the properties PROPERTY_1 and PROPERTY_2). In other words, to specify the retrieval spec of an object it is sufficient to specify the retrieval spec for any of its ancestors.

On the other hand, if we specifically specify IMO_TYPE_B, it will override any specification of its ancestors (in this case, IMO_TYPE_A). For example, if we modify the RS as follows:

```
<param name="oid">
  <value>{Some-object-of-type-B}}</value>
</param>
<param name="RS4">
<value>
<key name="RS_name">
  <key name="RequiredProperties">
    <key name=IMO_INTERFACE_A>
      <entry name=PROPERTY_1/>
      <entry name=PROPERTY_2/>
    </key>
    <key name=IMO_INTERFACE_B>
      <entry name=PROPERTY_3/>
    </key>
  </key>
</key>
</value>
</param>
```

In this case, we will get in the result IMO only PROPERTY_3, although its ancestor (IMO_TYPE_A) specified in the retrieval spec also PROPERTY_1 and PROPERTY_2. This means that specifying a retrieval spec for an interface overrides any retrieval spec defined for its ancestors.

The inheritance schemes enable us to simplify and generalize the RS. For example, in the case of CTPs, it is enough to specify the base type `IConnectionTerminationPoint` in the RS. This ensures that the whole CTP containment chain (all network layers) will be retrieved.

The specific details and inheritance scheme of all IMO objects are described in the IMO reference manual.

3.5.1.3 Excluding IMO Types and Properties

Similarly to the `requiredProperties` key, which specifies which IMO types and properties to include in the answer, there is the key `excludedProperties`, which describes what properties to exclude from the answer. This modifier is useful for filtering unneeded properties from the answer. For example:

```
<param name="oid">
  <value>{[ManagedElement(Key=Fore251Lab)]}</value>
</param>
<param name="rs">
  <value>
    <key name="RS2">
      <key name="requiredProperties">
        <key name="com.sheer.imo.IManagedElement">
          <entry name="*" />
        </key>
      </key>
      <key name="excludedProperties">
        <key name="com.sheer.imo.IManagedElement">
          <entry name="PhysicalRoot" />
        </key>
      </key>
    </key>
  </value>
</param>
```

In this case, the whole `IManagedElement` object will be retrieved, including all properties, except for `PhysicalRoot`.

The include and exclude sections are AND-ed, i.e. if the same property appears both in the `requiredProperties` and in the `excludeProperties` sections, the `excludeProperties` will prevail and the property will be discarded.

3.5.2 Format/Syntax

The XML layout of the Retrieval Specification is as follows

```
<param name="rs">
<value>
<key name="[rs name]">
  <entry name="register">[true/false]</entry>
  <key name="requiredProperties">
    <key name=[* or IMO type]>
      <entry name=[* or property name]/>
      <entry name=[* or property name]/>
      . . .
    </key>
  </key>
  <key name="excludedProperties">
    <key name=[* or IMO type]>
      <entry name=[* or property name]/>
      <entry name=[* or property name]/>
      . . .
    </key>
  </key>
  <key name="requiredAspects">
    <key name=[* or oid type]>
      <entry name=[* or aspect oid type]/>
      <entry name=[* or aspect oid type]/>
      . . .
    </key>
  </key>
  <key name="excludedAspects">
    <key name=[* or oid type]>
      <entry name=[* or aspect oid type]/>
      <entry name=[* or aspect oid type]/>
      . . .
    </key>
  </key>
</key>
</value>
</param>
```

- The name of the retrieval spec (`rs name`) is optional and used only for readability of the XML.
- The `register` entry indicates whether to register for changes on the objects that match the retrieval spec. This flag is optional and can be omitted. If not specified it is assumed to be false. See section 3.6.2 for description of registrations and notifications.
- The `requiredProperties` key indicates properties to include in the result. The `excludedProperties` key indicates properties to exclude from the result. The `requiredAspects` indicates aspects to attach to the result. All these modifiers are optional and may be discarded from the retrieval spec.
- For included and excluded properties you may specify a property name or wildcard (“*”) to represent all properties.
- A property that is both required and excluded will be excluded.

3.6 Aspects

Aspects are persistent information from the DNA DB, such as alarm history or business information entities that can be dynamically attached to any IMO object. The Sheer DNA fabric models only auto-discovered network information, and therefore it does not maintain any historical or business information. Examples for such business information are administrative details of the configured subscribers (name, phone number), descriptive labels for network objects, alarm history etc. The aspects mechanism allows to couple network information with persistent information of this type.

We can attach aspects to every IMO object, by specifying the required Aspects in the `requiredAspects` and `excludedAspects` sections in the RS. For example:

```
<command name="Get">
<param name="oid">
  <value>{[Snapshot]}</value>
</param>
<param name="rs">
  <value>
    <key name="RS5">
      <entry name="\register\ ">false</entry>
      <key name="\requiredProperties\ ">
        <key name="\com.sheer.imo.IManagedElement\ ">
          <entry name="\*\ "/>
        </key>
        <key name="\com.sheer.imo.IAlarmList\ ">
          <entry name="\*\ "/>
        </key>
        <key name="\com.sheer.imo.ICompiledAlarm\ ">
          <entry name="\*\ "/>
        </key>
      </key>
      <key name="excludedProperties">
        <key name="com.sheer.imo.topology.ISnapshot">
          <entry name="Links"/>
        </key>
      </key>
      <key name="requiredAspects">
        <key name="com.sheer.imo.keys.IManagedElementOid">
          <entry name="com.sheer.imo.keys.IAlarmListOid"/>
        </key>
      </key>
    </value>
  </param>
</command>
```

This GET command retrieves the network snapshot, excluding links (i.e., retrieves the list of devices). The aspects section of the retrieval specification is given by

```
<key name="requiredAspects">
  <key name="com.sheer.imo.keys.IManagedElementOid">
    <entry name="com.sheer.imo.keys.IAlarmListOid"/>
  </key>
</key>
```

This aspects specification specifies the GET command to attach to each returned `ManagedElement` IMO object the list of its corresponding alarms. The alarm list for each NE in the snapshot is an example of information that resides in the DNA DB, and cannot be inferred from the network.

The Aspects section in the retrieval spec is processed as follows:

1. The DNA system executes the GET query with the given retrieval spec while ignoring the Aspects. The query generates an IMO construct that contains all information that matches the retrieval spec.
2. DNA then inspects each IMO in the result, including IMOs that are properties of other IMOs. For each such IMO it checks if its OID matches any of the OIDs in the Aspects section in the retrieval spec. If so, it executes an internal GET command for the respective Aspects.
3. DNA then attaches all Aspects to their respective IMO object (according to the OID) and returns the combined output. The Aspects are attached in the XML response under a key called `Aspects` of type `IMObjects_array` (even if the array contains only one element)

3.6.1 Example

By executing the command in the previous section, we shall receive the following reply, which is a list of the NEs in the network, each with its related alarms (in this case we have one NE):

```
<IManagedElement>
  <ID type="Oid">{[ManagedElement(Key=ASAMSim)]}</ID>
  <CommunicationStateEnum type="Integer">3
    </CommunicationStateEnum>
  <DeviceName type="String">MINIRAMSim</DeviceName>
  <ElementCategoryEnum type="Integer">1</ElementCategoryEnum>
  <ElementTypeEnum type="Integer">27</ElementTypeEnum>
  <InvestigationStateEnum type="Integer">2
    </InvestigationStateEnum>
  <SysContact type="String">Alcatel</SysContact>
  <SysDescription type="String">ASAM</SysDescription>
  <SysLocation type="String">Alcatel Bell-Antwerpen
    </SysLocation>
  <SysName type="String">ASAM</SysName>
  <SysUpTime type="java.util.Date">
    Mon Feb 03 11:32:28 IST 2003</SysUpTime>
  <VendorEnum type="Integer">2</VendorEnum>
</IManagedElement>
<Aspects type="IMObjects_Array">
  <IAlarmList>
    <ID type="Oid">{[ManagedElement(Key=ASAMSim)]
      [AlarmList]}</ID>
    <Alarms type="IMObjects_Array">
      <ICompiledAlarm>
        <ID type="Oid">{[CompiledAlarm(Id=969)]}</ID>
        <AckStatusEnum type="Integer">0</AckStatusEnum>
        <ActiveAlarmCount type="Integer">3
          </ActiveAlarmCount>
        <AffectedSncNum type="Integer">53</AffectedSncNum>
        <AlarmType type="Integer">1</AlarmType>
        <CategoryEnum type="Integer">0</CategoryEnum>
        <Description type="">Null</Description>
        <EventTime type="java.util.Date">
          Mon Feb 03 17:05:47 IST 2003</EventTime>
      </ICompiledAlarm>
    </Alarms>
  </IAlarmList>
</Aspects>
</IManagedElement>
```

```

    <LatestModificationTime type="java.util.Date">
      Mon Feb 03 17:09:47 IST 2003
    </LatestModificationTime>
    <PerceivedSeverityEnum type="Integer">2
    </PerceivedSeverityEnum>
    <RemoveApproved type="Boolean">>false
    </RemoveApproved>
    <RootCause type="String">Link Up</RootCause>
    <Source type="Oid">
      {[TopologicalLink(LinkIdentifier=10.10.12.14-
        10.10.12.22_168430606_5365972917668516095_
        168430614_5365493598785573887_bi_)]}</Source>
    </ICompiledAlarm>
  </Alarms>
</IAlarmList>
</Aspects>

```

3.6.2 Specifying Properties vs. Aspects

The Aspects section is used only for the attachment of the DB object OID to the Network IMO object. The actual data contents of the DB IMO construct is specified in the `requiredProperties` section (same as for the network IMO objects). In the above example, we can see that in the `requiredAspects` section we extended the `ManagedElement` IMO object (the NE) to include the DB IMO object of type `IAlarmListOid` (which is the OID of the alarm container). However, in order to retrieve the actual alarms information, we have to specify it in the `requiredProperties` section, as follows:

```

    <key name=\"com.sheer.imo.IAlarmList\">
      <entry name=\"*\"/>
    </key>
    <key name=\"com.sheer.imo.ICompiledAlarm\">
      <entry name=\"*\"/>
    </key>

```

This instructs BQL to retrieve the `AlarmList` container, and all of its contained alarm objects (of type `ICompiledAlarm`).

3.6.3 Required Aspects vs. Excluded Aspects

As described above, the section is used only for defining the attachment of DB objects to network IMO objects, whereas the data retrieval scope in these DB objects is still specified in the `requiredProperties` section. Similarly, the `excludedAspects` section is used only for fine-tuning the attachment definition, and not for specifying the retrieved data scope. An example of the relationship between `requiredAspects` and `excludedAspects`, can be demonstrated in the case of IMO inheritance. Assume we have a generic DB IMO type A, and an inherited (more-specific) IMO type B, which is derived from A. We can specify the attachment of objects of type A in the `requiredAspects` section, while specifying object type B in the `excludedAspects` section. The result will be that all objects of type A (and their derivatives) will be attached, *except* for objects of type B (and their derivatives).

3.7 Registrations and Notifications

3.7.1 Registering for Notifications

When setting the `register` flag in the retrieval spec to “true”, the system returns the IMO construct that matches this retrieval spec and registers for changes on this data. I.e., registering for changes always performed as “Get and Register”. Any change detected in any of the data items will be notified in an unsolicited message to the BQL client.

When the BQL session is closed, the system clears all registrations associated with this session. Therefore, there is no need to explicitly unregister before closing the BQL session.

When registering for notifications, the user may optionally specify a registration ID for each registration. This registration ID may be used later for unregistering. Note that the registration ID should be unique and it is up to the client application to ensure uniqueness. If the user performs two registrations with the same ID the second registration will override the first one. The first registration will still be active and will send notifications, but the user will not be able to unregister it. In such case, the only way to unregister the first registration is by issuing an “unregister all” command.

To specify a command ID in a register command append `commandId=ID` to the beginning of the GET command. For example:

```
commandId=5
<command name="Get">
  <param name="oid">
    <value>{ [ManagedElement(Key=RB1800Sim)] }</value>
  </param>
  <param name="rs">
    <value>
      <key name="RS6">
        <entry name="register">true</entry>
        <key name="requiredProperties">
          <key name="com.sheer.imo.ICompiledAlarm">
            <entry name="*" />
          </key>
        </key>
      </key>
    </value>
  </param>
</command>
```

3.7.2 Unregistering

To unregister a specific registration ID send an unregister command with the command ID as a parameter. For example:

```
<command name="unregister">
  <param name="commandId">
    <value>5</value>
  </param>
</commands>
```

To unregister from all previous registrations use “all” as the command ID:

```
<command name="unregister">
  <param name="commandId">
    <value>all</value>
  </param>
</commands>
```

When the system detects a change in a registered object it responds with an unsolicited notification message, which are described in the next section.

3.7.3 Notifications

A notification is an IMO object that describes the change in another IMO. For example:

```
commandId=7
<IMObjects_Array>
  <IScalarNotification>
    <ID type="Oid">{[Notification(SequenceNumber=249)
      (Time=1052903741029)]}</ID>
    <PropertyName type="String">RemoveApproved</PropertyName>
    <NewIMO type="ICompiledAlarm">
      <ID type="Oid">{[CompiledAlarm(Id=7)]}</ID>
      <RemoveApproved type="Boolean">true</RemoveApproved>
    </NewIMO>
    <OldIMO type="ICompiledAlarm">
      <ID type="Oid">{[ManagedElement(Key=RB1800Sim)
        [AlarmList][CompiledAlarm(Id=7)]}</ID>
      <RemoveApproved type="Boolean">false</RemoveApproved>
    </OldIMO>
  </IScalarNotification>
</IMObjects_Array>
```

In this example the notification describes that the in an IMO object of type `ICompiledAlarm` the property `RemoveApproved` changed its state from “false” to “true”. The notification is always of type `IMObjects_Array` and contains an IMO object for each notification. In this example it contains only a single notification of type `IScalarNotification`, which indicates a change in a single property. The notification includes the registration ID, and (for each change object) the type of the changed property, the time & sequence number, and the new & old values.

3.7.3.1 IScalarNotification

`IScalarNotification` describes changes in one or more properties of an IMO object. Consider the previous example:

```
commandId=7
<IMObjects_Array>
  <IScalarNotification>
    <ID type="Oid">{[Notification(SequenceNumber=249)
      (Time=1052903741029)]}</ID>
    <PropertyName type="String">RemoveApproved</PropertyName>
    <NewIMO type="ICompiledAlarm">
      <ID type="Oid">{[CompiledAlarm(Id=7)]}</ID>
      <RemoveApproved type="Boolean">true</RemoveApproved>
    </NewIMO>
    <OldIMO type="ICompiledAlarm">
      <ID type="Oid">{[ManagedElement(Key=RB1800Sim)
        [AlarmList][CompiledAlarm(Id=7)]]}</ID>
      <RemoveApproved type="Boolean">false</RemoveApproved>
    </OldIMO>
  </IScalarNotification>
</IMObjects_Array>
```

The notification is comprised of four elements:

- `Oid` The OID of the notification. Identifies the sequence number and the time of the notification.
- `NewIMO` The IMO object after the change
- `OldIMO` The IMO object before the change
- `PropertyName` The property in the IMO that was changed

In the example above, the scalar notification describes a change in the property `RemoveApproved` for a specific alarm of the device RB1800Sim.

3.7.3.2 IAddNotification

`IAddNotification` describes that an IMO object was added to the system.

For example:

```
commandId=alarms1
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IAddNotification>
    <ID type="Oid">{[Notification(SequenceNumber=253)
      (Time=1052903750795)]}</ID>
    <PropertyName type="String">Alarms</PropertyName>
    <NewIMO type="IAlarmList">
      <ID type="Oid">{[ManagedElement(Key=RB1800Sim)
        [AlarmList]]}</ID>
      <Alarms type="IMObjects_Array">
        <ICompiledAlarm>
          <ID type="Oid">{[CompiledAlarm(Id=7)]}</ID>
        </ICompiledAlarm>
      </Alarms>
    </NewIMO>
  </ IAddNotification >
</IMObjects_Array>
```

This example describes that a `CompiledAlarm` with id 7 was added to the Alarms table.

- `Oid` OID of the notification. Identifies the sequence number and the time of the notification
- `NewIMO` The added IMO
- `PropertyName` The table whose element was added. In this example it states that the “Alarms” table changed. The change is the addition of a compiled alarm.

3.7.3.3 IRemoveNotification

`IRemoveNotification` is sent by a containing object, to notify that one of its internal IMO objects was deleted. For example:

```
commandId=alarms1
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IRemoveNotification>
    <ID type="Oid">{[Notification(SequenceNumber=253)
      (Time=1052903750795)]}</ID>
    <PropertyName type="String">Alarms</PropertyName>
    <NewIMO type="IAlarmList">
      <ID type="Oid">{[ManagedElement(Key=RB1800Sim)]
        [AlarmList]}</ID>
      <Alarms type="IMObjects_Array">
        <ICompiledAlarm>
          <ID type="Oid">{[CompiledAlarm(Id=7)]}</ID>
        </ICompiledAlarm>
      </Alarms>
    </NewIMO>
  </IRemoveNotification>
</IMObjects_Array>
```

The OID of the removed element is always given under the `NewIMO` key.

The notification `IRemoveNotification` is comprised of three elements:

- `Oid` OID of the notification. Identifies the sequence number and the time of the notification
- `NewIMO` The removed IMO
- `PropertyName` The containing object whose element was deleted. In this example it states that the “Alarms” table changed. The change is the removal of a compiled alarm.

3.7.3.4 IObjectDeleteNotification

`IObjectDeleteNotification` is sent by an object when it is deleted. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IObjectDeleteNotification>
    <ID type="Oid">{[Notification(SequenceNumber=1201)
      (Time=1052837316305)]}</ID>
    <Oid type="Oid">{[ManagedElement(Key=RB1)][LogicalRoot]
      [Context(ContextName=cyber.net)]}</Oid>
  </IObjectDeleteNotification>
</IMObjects_Array>
```

This example demonstrates the notification in the case that a context was deleted from the NE named RB1. `IObjectDeletedNotification` reports the OID of the deleted element.

Note the difference between `IRemoveNotification` and `IObjectDeletedNotification`. `IRemoveNotification` is generated for a containing object when one of its elements is removed. For example, when registering on a VC, we shall receive this notification from the table anytime a VC is removed. However, if we would have registered on the specific VC, we would receive an `IObjectDeletedNotification` notification when the VC is removed.

4 BQL Generic Commands

4.1 Introduction

BQL defines five generic commands, which are supported by most IMO objects. These commands are:

- **Get** – Retrieve an information for a specific object
- **Update** – Set properties & relations to existing objects
- **Create** – Create a new IMO object
- **Delete** – Delete an existing IMO object
- **Find** – Retrieve information for several objects according to search criteria.

In this section we describe each of these generic commands. There are other specialized commands that allow performing operations that cannot be mapped into one of these basic commands, e.g. Activation commands. These specialized commands are beyond the scope of this manual.

Since Sheer DNA is based on network auto-discovery, the Update, Create and Delete commands may be invalid for some IMO objects. Moreover, not all IMO objects support all the generic commands. The supported commands for each IMO are described in the IMO reference manual.

4.2 GET Command

4.2.1 Description

The GET command retrieves an IMO construct (related IMO objects) based on a specified retrieval specification (RS). It also enables registering for notifications on changes in the specified IMO objects.

4.2.2 Syntax

```
GET_COMMAND:      <command name="Get">
                   <param name="oid">
                     <value> OID </value>
                   </param>
                   <param name="rs">
                     <value>
                       RETREIVAL_SPEC
                     </value>
                   </param>
                 </command>

OID:              OID format as explained in section
                 3.2.1

RETREIVAL_SPEC:  Retrieval spec format as explained in
                 section 3.5.2
```

4.2.3 Output

The output is IMO data in XML format. If the GET command registers for changes then the output would terminate with `\n$\n`. Otherwise, the output of the command will end with the standard EOT sequence `\n.\n`.

4.2.4 Example

```
<command name="Get">
  <param name="oid">
    <value>{ [ManagedElement(Key=ASAMSim)] }</value>
  </param>
  <param name="rs">
    <value>
      <key name="">
        <entry name="register">false</entry>
        <key name="requiredProperties">
          <key name="com.sheer.imo.IManagedElement">
            <entry name="*" />
          </key>
        </key>
      </key>
    </value>
  </param>
</command>
```

4.3 UPDATE Command

4.3.1 Description

The UPDATE command modifies an existing IMO object.

The command receives an OID of an object to modify and an array of notifications. The notifications are of type scalar notification, add notification and remove notification. These notifications describe how to modify the IMO object, i.e., what properties to change, add or remove.

4.3.2 Syntax

```
UPDATE_COMMAND:      <command name="Update">
                      <param name="oid">
                        <value> OID </value>
                      </param>
                      <param name="imoobjectArr">
                        <value>
                          NOTIFICATION_ARRAY
                        </value>
                      </param>
                    </command>

OID:                  OID format as explained in
                      section 3.2.1

NOTIFICATION_ARRAY:  NOTIFICATION_IMO |
                      NOTIFICATION_IMO
                      NOTIFICATION_ARRAY

NOTIFICATION_OBJECT: IMO object of type
                      IScalarNotification,
                      IAddNotification or
                      IRemoveNotification as described
                      in section 3.7.3
```

4.3.3 Output

None

4.3.4 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<command name="Update">
  <param name="oid">
    <value>
      {[ManagedElement(Key=ACESim)][BusinessObject]}
    </value>
  </param>
  <param name="imobjectArr">
    <value>
      <IScalarNotification>
        <ID type="Oid">{[Notification]}</ID>
        <PropertyName type="String">EKey</PropertyName>
        <NewIMO type="IBusinessObject">
          <ID type="Oid">{[ManagedElement(Key=ACESim)]
            [BusinessObject]}</ID>
          <EKey type="String">key2</EKey>
          <Name type="String">name2</Name>
        </NewIMO>
      </IScalarNotification>
    </value>
    <value>
      <IScalarNotification>
        <ID type="Oid">{[Notification]}</ID>
        <PropertyName type="String">Name</PropertyName>
        <NewIMO type="IBusinessObject">
          <ID type="Oid">{[ManagedElement(Key=ACESim)]
            [BusinessObject]}</ID>
          <EKey type="String">key2</EKey>
          <Name type="String">name2</Name>
        </NewIMO>
      </IScalarNotification>
    </value>
  </param>
</command>
```

4.4 CREATE Command

4.4.1 Description

The create command creates a new data object in the system. The command receives as a parameter the data of the new IMO object.

4.4.2 Syntax

```
CREATE_COMMAND:    <command name="Create">
                   <param name="imobject">
                     <value>
                       IMO_DATA
                     </value>
                   </param>
                 </command>

IMO_DATA:          IMO object representation
```

4.4.3 Output

None

4.4.4 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<command name="Create">
  <param name="imobject">
    <value>
      <IBusinessObject>
        <ID type="Oid">{[ManagedElement(Key=ACESim)]
          [BusinessObject]}

```

4.5 DELETE Command

4.5.1 Description

The Delete command deletes existing IMO object. The deleted element is identified by its OID. Note that not all data elements in the system can be deleted.

4.5.2 Syntax

```
DELETE_COMMAND:    <command name="Delete">
                    <param name="oid">
                        <value> OID </value>
                    </param>
                </command>

OID:                OID format as explained in section
                    3.2.1
```

4.5.3 Output

None

4.5.4 Example

```
<?xml version="1.0" encoding="UTF-8"?>
<command name="Delete">
  <param name="oid">
    <value>
      {[ManagedElement(Key=ACESim)][BusinessObject]}
    </value>
  </param>
</command>
```

4.6 Find

4.6.1 Description

The “Find” command retrieves a list of data entities (IMO objects) according to a search qualifier. The command includes a Retrieval Spec and an IMO object which serves as a search qualifier.

The `Find` section specifies a list of IMO objects and their respective qualifier values. For example, if our RS is for port information, we can specify a qualifier value for the port `OperStatus`, and the reply will only contain those ports that have this status.

4.6.2 Syntax

```
FIND_COMMAND:      <command name="Find">
                    <param name="imo">
                      <value> IMO </value>
                    </param>
                    <param name="rs">
                      <value>
                        RETRIEVAL_SPEC
                      </value>
                    </param>
                  </command>

IMO:               This IMO object defines the search
                  qualifier. The search qualifier
                  object includes one or more
                  properties with specified values.

RETRIEVAL_SPEC:   Retrieval spec format as explained in
                  section 3.5.2. Defines what is
                  returned from the objects that match
                  the search criteria.
```

4.6.3 Output

The function returns the IMO objects (of the same type as the search qualifier) whose properties match the properties of the search qualifier. For each such object it returns its part that matches the retrieval spec. The output is `IMObject_Array` where each element in the array is the IMO that matches the retrieval spec.

For example, to find subscribers named “John Smith”, we should:

Specify as search qualifier a *Business Attachment* IMO object, in which name = “*John Smith*” and `typeEnum = 1` (subscriber)

Provide a Retrieval Spec for Business Attachment object types.

4.6.4 Example

```
<command name="Find">
  <param name="imo">
    <value>
      <IBusinessObject>
        <Name type="String">John Smith</Name>
        <TypeEnum type="Integer">1</TypeEnum>
      </IBusinessObject>
    </value>
  </param>
  <param name="rs">
    <value>
      <key name="find-business-object">
        <entry name="register">>false</entry>
        <key name="requiredProperties">
          <key name=
            "com.sheer.imo.IBusinessObject">
            <entry name="*" />
          </key>
        </key>
      </key>
    </value>
  </param>
</command>
```

The output of this query will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<IMObjects_Array>
  <IBusinessObject>
    <ID type="Oid">{[ManagedElement(Key=ACESim)]
      [BusinessObject]}</ID>
    <EKey type="String">key2</EKey>
    <Name type="String">name2</Name>
    <Notes type="String"/>
    <TypeEnum type="Integer">0</TypeEnum>
  </IBusinessObject>
</IMObjects_Array>
```

4.7 Refresh

4.7.1 Description

The IMO model polls its data from the VNE model, the VNE model is polling its data from the devices. When a "Get" command is executed, a set of IMOs is retrieved. Sometimes the data is not updated to the current device state as it was polled from the VNE model. When the most recent data is needed, use the Refresh command to request the VNE to "retrieve now" the data from the device before polling its model.

4.7.2 Example

```
<command name="Refresh">
  <param name="oid">
    <value>{[ManagedElement(Key=C_1)]}</value>
  </param>
  <param name="oids">
    <value>{[ManagedElement(Key=C_1)][PhysicalRoot][Chassis]
      [Slot(SlotNum=0)][Module]
      [Port(PortNumber=FastEthernet0/0)]}</value>
    <value>{[ManagedElement(Key=C_1)][PhysicalRoot][Chassis]
      [Slot(SlotNum=0)][Module]
      [Port(PortNumber=FastEthernet0/0)]
      [PhysicalLayer]}</value>
    <value>{[ManagedElement(Key=C_1)][PhysicalRoot][Chassis]
      [Slot(SlotNum=0)][Module]
      [Port(PortNumber=FastEthernet0/0)]
      [PhysicalLayer][DataLinkLayer]}</value>
    <value>{[ManagedElement(Key=C_1)][PhysicalRoot][Chassis]
      [Slot(SlotNum=0)][Module]
      [Port(PortNumber=FastEthernet0/0)]
      PhysicalLayer][DataLinkLayer][VlanEncapMux]}</value>
  </param>
</command>
```

