ExtremeXOS® InSite Quick Start Guide

Extreme Networks, Inc.
3585 Monroe Street
Santa Clara, California 95051
(888) 257-3000
(408) 579-2800
http://www.extremenetworks.com

Published: March 2010
Part number: 120276-00 Rev 02

Table of Contents

1 Getting Started

This document includes the following sections:

Overview

This guide is intended as a quick start reference for ExtremeXOS® InSite, a software development kit (SDK) that extends the capabilities of ExtremeXOS through a set of comprehensive application programming interfaces (APIs). These interfaces augment the networking power of ExtremeXOS by allowing applications the ability to interface directly with Extreme Networks® switches.

Extreme Networks XML APIs enable reliable and secure external device-to-device management communication. The API interface provides a mechanism to communicate with Extreme Networks switches using XML messages. Its standards-based SOAP/XML architecture makes it easy to integrate the network infrastructure with higher-level application and business software. The configuration and monitoring capabilities provided by the APIs let you create Service Oriented Architecture (SOA) solutions that bridge the gap between application and business logic with network configuration and events.

The ExtremeXOS InSite software development kit includes the following:

- *ExtremeXOS InSite Quick Start Guide*
- Web Services Description Language (WSDL) files that define operations or interfaces that are available and XML Schema Definitions (XSD) files that define the objects for the operations
- API reference documentation
- Example Java and Perl programs

XML Notification Client Support

The XML Notification Client feature is available from ExtremeXOS 12.4.1. This client can be enabled on all ExtremeXOS switches. Events (such as a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system) can be sent as an asynchronous message/event notification to external web servers. ExtremeXOS modules that support the XML notification are

Identity Manager and Event Management System (EMS). XML notification does not provide any event filtering capability. For details regarding XML Notification and Identity Manager commands refer to the following documents found on the Extreme Networks documentation website at http://www.extremenetworks.com/go/documentation

- *ExtremeXOS Concepts Guide*
- *ExtremeXOS Command Reference Guide*

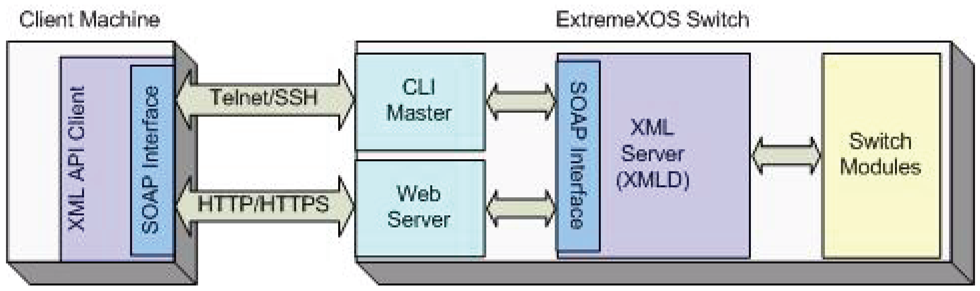
Audience

This guide is intended for software developers using a programming interface to write applications for Extreme Networks devices. It assumes a basic working knowledge of the following:

- Command line interface (CLI) for ExtremeXOS switches
- Concepts and operation of XML API interfaces and Web services
- Writing Web services client code in the programming language (such as, Java, C/C++, PERL) being used

Architecture

Figure 1: XML API Client Interface to an XML Server



The XML server (XMLD) shown in Figure 1 is responsible for providing a gateway between the external interface and the switch modules. It enforces security; wraps, unwraps, and validates messages; and performs the mechanical translations of results from the modules to the client machine.

The XML APIs use the SOAP protocol over telnet/SSH or HTTP/HTTPS to exchange XML configuration messages between the client machine and the ExtremeXOS switch modules.

By describing the XML API in WSDL and the interface through SOAP, developers can leverage existing public WSDL tools. For example, Apache Axis or Perl SOAP::Lite modules can be used to generate code automatically to build management applications.

WSDL documents describe the Web services used. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. These style/use models are RPC/encoded, RPC/literal, document/encoded, and document/literal. There is one additional pattern called the document/literal wrapped pattern. WSDLs for ExtremeXOS XML APIs use the document/literal and the document/literal wrapped pattern models.

Transport Protocols

You can use either a telnet/SSH or an HTTP/HTTPS transport protocol to interface to the XML server.

XML Over Telnet/SSH

This protocol provides a mechanism to use the APIs by sending XML messages over an SSH or telnet session. Once an SSH or telnet connection is established with the switch, the client must first set the session to XML mode before using the XML APIs by sending the following CLI command to the switch:

```
enable xml-mode
```

The session can now be used to send and receive XML messages.

XML Over HTTP/HTTPS

The XML APIs also support HTTP/HTTPS as a protocol. In this case, the client sends XML messages as SOAP packets to the Web server on the switch. If the Web server is not running on the switch, it can be started by entering one of the following commands:

```
enable web http
enable web https
```

The binding address for the webservice is

- `http://<device>/xmlService for HTTP`
- `https://<device>/xmlService for HTTPS`

  where <device> is the IP address or the host name of the switch.

NOTE

The Secure Sockets Layer (SSL) module must be installed before using HTTPS. See the ExtremeXOS Concepts Guide *for details about using SSL on the switch.*

System Requirements

The following are switch and client hardware and software requirements.

On the Switch

The hardware interface requires an ExtremeXOS-based switch that is up and running. If you are using XML over HTTP/HTTPS, Web access must be enabled.

On the XML API Client

The ExtremeXOS InSite client setup requires the following:

- ExtremeXOS InSite software development kit
- Developer tools for Web services programming, such as:
  - gSoap for C/C++ programming
  - SOAP:Lite for Perl programming
  - Apache Axis for Java programming
  - Flex programming

Object-Based APIs

Object-based APIs are shown in Table 1. These APIs handle ports, VLANs, stacking, inventory, port statistics, user accounts, RADIUS management, TACACS management, session management, identity management, and Provider Backbone Bridge (PBB).

NOTE

Identity Manager is not supported on the BlackDiamond® 20808 switch. PBB XML APIs are supported on the BlackDiamond 20808 switch.

These APIs define the four basic operations—*get*, *create*, *set*, *delete*. The operations can be used on objects that are derived from the abstract ExosBase object. For example, to retrieve information about the slots on a switch, the *get* operation is used with the SlotInfo object. The SlotInfo object is derived from an ExosBase object and represents a slot on the switch. It is passed in as the filter parameter for the *get* operation.

Client applications will use the *openSession* operation to authenticate and open a session on the switch. A reference ID for the session is passed back to the client application. The client must use this sessionid for all the operations. It is passed as part of header information either in the SOAP:Header or SOAP:Body of the request.

Table 1: Object-Based APIs

Operation	Description
get	Gets one or more objects from a switch depending on the filter.
set	Sets an object on a switch depending on the filter.
create	Creates a new object on a switch.
delete	Deletes an existing object on a switch.
openSession	Opens a session on a switch.
closeSession	Terminates a session on a switch.
keepAlive	Keeps a session alive without being timed out.

Interpreting Schema Annotation

The schema files use the annotation section to provide a description of each element. The description also includes the following information for the element.

- *Version* indicates the ExtremeXOS release when support was added for the element. To use an XML API client written for an older version of ExtremeXOS with a newer version of ExtremeXOS, the client can specify the specific version in the open session calls.
- *Access* indicates the access type of the element.
  - *READ*—element can be queried using a *get* request, but it cannot be modified.
  - *WRITE*—element can be configured using a *set* request.
  - *INDEX*—element is an index parameter.

For example, the following is the definition of the portList and adminState elements in port.xsd.

```
<xsd:element name="portList" type="common:StringLen8" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>
      portlist variable represents port id. This is a required element.
        Version: EXOS 12.0. Access: READ, INDEX.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="adminState" type="AdminStates" minOccurs="0">
  <xsd:annotation>
    <xsd:documentation>
     Version: EXOS 12.0. Access: READ, WRITE.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

The version information indicates that support for these elements was added in ExtremeXOS release 12.0. The READ access type for portList indicates that it can be queried but cannot be modified. The READ, WRITE access type for adminState indicates that it can be queried and modified.

Error Handling and Error Messages

If the XML API fails, the API can throw a checked exception or return data and errors depending on the operation and the severity. Checked exceptions can be defined in the XML API definition so the client knows the error format to expect from the server. This exception has a fault code and a message that provides information about the failure.

The following example shows a system response returned with an error.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
.
. (Namespace listings...)
.
   <SOAP-ENV:Body>
      <SOAP-ENV:Fault>
         <faultcode>SOAP-ENV:Client</faultcode>
         <faultstring>Invalid session id</faultstring>
      </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Programming with Object-Based APIs

This section describes a typical object-based client/switch request/response interchange using the Java programming language and SOAP protocol. The XML API client must authenticate with the ExtremeXOS switch to gain authorization, perform the proper handshake, and keep the session alive to avoid timing out. The XML APIs use the same user name and password as the ExtremeXOS command line interface (CLI), so the same user privilege levels are applied to all requests coming from the XML interface. If the client is not an admin user type, any XML request to modify the switch configuration is rejected.

Opening a Session

The XML API client logs in using the open session request call. In this message, the client can send the following information about its application.

Table 2: Open Session Information

Item	Description
appName	Name of the client application
username	Username used to log in to the switch
password	Password used to log in to the switch
xmlApiVersion	Version of the API with which the client was written. This is the mechanism used by the switch to determine client capabilities.
sessionId	Session id obtained after a successful openSession request.
timeout	Idle timeout value in seconds. The switch will close sessions that are idle for more than this period. The default timeout is 900 seconds. Client can choose to specify a different timeout value or use keepAlive requests to maintain a session during periods of inactivity. NOTE: If the usage pattern for the client has long periods of inactivity between calls, it may be more efficient to close the session and create a new one.
accessRight	Access privileges for the username used to log in to the switch.

Send an open session request with your user name and password. If authenticated, a sessionId is returned. The client machine provides its application name, user name, password, and the version number of the XML APIs it intends to use.

```
Session session=new Session();
session.setUsername(“user1”);
session.setPassword(“mypassword”);
session.setAppName(“API Test Client");
```

User name and password for authentication

```
OpenSessionRequest req=new OpenSessionRequest();
req.setSession(session);

SwitchPortType stub=getSwitchStub(device);
OpenSessionReply reply=stub.openSession(req);
```

Open a session

Following is a snippet of the SOAP envelope contents of the open session request.

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <openSessionRequest
         xmlns="http://www.extremenetworks.com/XMLSchema/xos/common">
         <session xmlns="">
            <appName>API Test Client</appName>
            <username>sqa</username>
            <password></password>
         </session>
      </openSessionRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Getting the System Response

```
String
sessionId=reply.getSession().getSessionId();
```

A sessionId returned from the switch

The system response confirms the application name, user name, password, and version number, then supplies a sessionId to be used by the client for all exchanges for this session, a maximum idle timeout length, and the recognized access rights of the user. The default timeout is 15 minutes (900 seconds). A keep-alive message must be sent periodically to prevent the server from terminating the session.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ns1="http://www.extremenetworks.com/XMLSchema/xos/l2protocol"
   xmlns:xos="urn:xapi"
   xmlns:vlan="http://www.extremenetworks.com/XMLSchema/xos/vlan"
```

```
  xmlns:port="http://www.extremenetworks.com/XMLSchema/xos/port"
  xmlns:ns2="http://www.extremenetworks.com/XMLSchema/xos/dhcp"
  xmlns:aaa="http://www.extremenetworks.com/XMLSchema/xos/aaa"
  xmlns:snmp="http://www.extremenetworks.com/XMLSchema/xos/snmp"
  xmlns:system="http://www.extremenetworks.com/XMLSchema/xos/system"
  xmlns:ns4="urn:ietf:params:xml:ns:netconf:soap:1.0"
  xmlns:netb="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:switch="http://www.extremenetworks.com/XMLSchema/xos/switch"
  xmlns:com="http://www.extremenetworks.com/XMLSchema/xos/common"
  xmlns:upm="http://www.extremenetworks.com/XMLSchema/xos/upm"
  xmlns:xosacl="urn:xapi/l2protocol/acl"
  xmlns:xoscfg="urn:xapi/cfgmgmt/cfgmgr"
  xmlns:xosfdb="urn:xapi/l2protocol/fdb"
  xmlns:xospol="urn:xapi/system/policy"
  xmlns:xosvlan="urn:xapi/l2protocol/vlan">
  <SOAP-ENV:Body>
    <com:openSessionReply>
      <session>
        <appName>API Test Client</appName>
        <username>sqa</username>
        <password />
        <xmlApiVersion>12.1.0.54</xmlApiVersion>
        <sessionId>25af0000000028</sessionId>
        <timeout>900</timeout>
        <accessRight>readWrite</accessRight>
      </session>
    </com:openSessionReply>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Making a Get Request

Send the API request using the sessionId obtained from the system response. The example code queries the inventory module to get information about all the slots in the system.

```
ClientHeader hdr=new ClientHeader();
hdr.setSessionId(sessionId);
```

Use sessionId provided by the system.

```
SlotInfo slotInfo = new SlotInfo();
GetRequest req = new GetRequest();
req.setHdr(hdr);
req.setFilter(slotInfo);

SwitchPortType stub=getSwitchStub(device);
GetResponse resp=stub.get(req);
```

Send the request to get all slot information

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
```

```
      <ns1:hdr
         xmlns:ns1="http://www.extremenetworks.com/XMLSchema/xos/common">
         <reqId>1</reqId>
         <sessionId>25af0000000028</sessionId>
      </ns1:hdr>
   </soapenv:Header>
   <soapenv:Body>
      <getRequest maxSize="0"
         xmlns="http://www.extremenetworks.com/XMLSchema/xos/switch">
         <filter xsi:type="ns2:SlotInfo" xmlns=""
            xmlns:ns2="http://www.extremenetworks.com/XMLSchema/xos/system" />
      </getRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Getting the Requested Data

The switch replies with the requested data.

```
ExosBase temp[] = resp.getObjects();
for(int i = 0; i < temp.length; i++ ) {
    SlotInfo slot= (SlotInfo)temp[i];
    System.out.println("Slot:
                "+slot.getSlotNumber() +
                 " Type : "+slot.getCardType());

 }
```

Data returned from the switch

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ns1="http://www.extremenetworks.com/XMLSchema/xos/l2protocol"
   xmlns:xos="urn:xapi"
   xmlns:vlan="http://www.extremenetworks.com/XMLSchema/xos/vlan"
   xmlns:port="http://www.extremenetworks.com/XMLSchema/xos/port"
   xmlns:ns2="http://www.extremenetworks.com/XMLSchema/xos/dhcp"
   xmlns:aaa="http://www.extremenetworks.com/XMLSchema/xos/aaa"
   xmlns:snmp="http://www.extremenetworks.com/XMLSchema/xos/snmp"
   xmlns:system="http://www.extremenetworks.com/XMLSchema/xos/system"
   xmlns:ns4="urn:ietf:params:xml:ns:netconf:soap:1.0"
   xmlns:netb="urn:ietf:params:xml:ns:netconf:base:1.0"
   xmlns:switch="http://www.extremenetworks.com/XMLSchema/xos/switch"
   xmlns:com="http://www.extremenetworks.com/XMLSchema/xos/common"
   xmlns:upm="http://www.extremenetworks.com/XMLSchema/xos/upm"
   xmlns:xosacl="urn:xapi/l2protocol/acl"
   xmlns:xoscfg="urn:xapi/cfgmgmt/cfgmgr"
   xmlns:xosfdb="urn:xapi/l2protocol/fdb"
   xmlns:xospol="urn:xapi/system/policy"
   xmlns:xosvlan="urn:xapi/l2protocol/vlan">
   <SOAP-ENV:Header>
```

```
      <com:hdr>
         <reqId>1</reqId>
         <sessionId>25af0000000028</sessionId>
      </com:hdr>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <switch:getResponse>
         <objects>
            <object xsi:type="system:SlotInfo">
               <slotNumber>1</slotNumber>
               <nameOfHw>Slot-1</nameOfHw>
               <cardType />
               <configuredCardType />
               <cardState>Empty</cardState>
               <nodeState />
               <swVersion />
               <swBuild />
               <numberOfPorts>0</numberOfPorts>
               <serialNumber />
               <flags />
               <lastError />
               <selectedConfig>NONE</selectedConfig>
               <imageInfo>
                  <primaryImage />
                  <secondaryImage />
                  <imageSelected />
                  <imageBooted />
               </imageInfo>
               <temperature />
               <bootRom />
               <odometer />
            </object>
            <object xsi:type="system:SlotInfo">
               <slotNumber>2</slotNumber>
               <nameOfHw>Slot-2</nameOfHw>
               <cardType>GM-20T</cardType>
               <configuredCardType>GM-20T</configuredCardType>
               <cardState>Operational</cardState>
               <nodeState />
               <swVersion />
               <swBuild />
               <numberOfPorts>20</numberOfPorts>
               <serialNumber>
                  804024-00-09 06154-00088
               </serialNumber>
               <flags>M</flags>
               <lastError />
               <selectedConfig>NONE</selectedConfig>
               <imageInfo>
                  <primaryImage />
                  <secondaryImage />
                  <imageSelected />
                  <imageBooted />
               </imageInfo>
               <temperature>30.00 Normal</temperature>
               <bootRom />
```

```
    <odometer>
      465 days 6 hours since Jun-23-2006
    </odometer>
  </object>
  <object xsi:type="system:SlotInfo">
    <slotNumber>5</slotNumber>
    <nameOfHw>Slot-5</nameOfHw>
    <cardType>XM-2X</cardType>
    <configuredCardType>XM-2X</configuredCardType>
    <cardState>Operational</cardState>
    <nodeState />
    <swVersion />
    <swBuild />
    <numberOfPorts>2</numberOfPorts>
    <serialNumber>
      804018-00-02 05384-00002
    </serialNumber>
    <flags>M</flags>
    <lastError />
    <selectedConfig>NONE</selectedConfig>
    <imageInfo>
      <primaryImage />
      <secondaryImage />
      <imageSelected />
      <imageBooted />
    </imageInfo>
    <temperature>36.00 Normal</temperature>
    <bootRom />
    <odometer>
      747 days 5 hours 30 minutes since Nov-18-2005
    </odometer>
  </object>
  <object xsi:type="system:SlotInfo">
    <slotNumber>6</slotNumber>
    <nameOfHw>Slot-6</nameOfHw>
    <cardType />
    <configuredCardType />
    <cardState>Empty</cardState>
    <nodeState />
    <swVersion />
    <swBuild />
    <numberOfPorts>0</numberOfPorts>
    <serialNumber />
    <flags />
    <lastError />
    <selectedConfig>NONE</selectedConfig>
    <imageInfo>
      <primaryImage />
      <secondaryImage />
      <imageSelected />
      <imageBooted />
    </imageInfo>
    <temperature />
    <bootRom />
    <odometer />
  </object>
```

```
            <object xsi:type="system:SlotInfo">
               <slotNumber>9</slotNumber>
               <nameOfHw>MSM-A</nameOfHw>
               <cardType>MSM-5</cardType>
               <configuredCardType />
               <cardState>Operational</cardState>
               <nodeState>MASTER</nodeState>
               <swVersion>12.1.0.54</swVersion>
               <swBuild />
               <numberOfPorts>0</numberOfPorts>
               <serialNumber>
                  804046-00-05 06314-00045
               </serialNumber>
               <flags />
               <lastError />
               <selectedConfig>primary.cfg</selectedConfig>
               <imageInfo>
                  <primaryImage>12.1.0.54</primaryImage>
                  <secondaryImage>12.0.2.12</secondaryImage>
                  <imageSelected>primary</imageSelected>
                  <imageBooted>primary</imageBooted>
               </imageInfo>
               <temperature>34.00 Normal</temperature>
               <bootRom>1.0.0.2</bootRom>
               <odometer>
                  438 days 11 hours since Aug-27-2006
               </odometer>
            </object>
            <object xsi:type="system:SlotInfo">
               <slotNumber>10</slotNumber>
               <nameOfHw>MSM-B</nameOfHw>
               <cardType />
               <configuredCardType />
               <cardState>Empty</cardState>
               <nodeState />
               <swVersion />
               <swBuild />
               <numberOfPorts>0</numberOfPorts>
               <serialNumber />
               <flags />
               <lastError />
               <selectedConfig>NONE</selectedConfig>
               <imageInfo>
                  <primaryImage />
                  <secondaryImage />
                  <imageSelected />
                  <imageBooted />
               </imageInfo>
               <temperature />
               <bootRom />
               <odometer />
            </object>
         </objects>
      </switch:getResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Closing a Session

Close the session by sending a request using the original sessionId. Sessions can also be closed manually using the CLI `clear session` command.

```
CloseSessionRequest req=new
CloseSessionRequest();
req.setSessionId(sessionId);

SwitchPortType stub=getSwitchStub(device);
CloseSessionReply reply=stub.closeSession(req);
```

Close a session

The client includes the sessionId in the close session request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:com="http://www.extremenetworks.com/XMLSchema/xos/common">

   <soapenv:Header/>
   <soapenv:Body>
      <com:closeSessionRequest>
          <sessionId>20af0000000005</sessionId>
      </com:closeSessionRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Function-Based APIs

Function-based APIs are described in the upm.wsdl and xos.wsdl files.

UPM APIs

The available UPM APIs are shown in Table 3. The APIs handle profiles and timers. The general steps (openSession, operation, closeSession) are the same as for Object-based APIs.

Table 3: UPM APIs Provided in the UPM.WSDL File

Operation	Description
bindProfile	Binds the profile to ports.
closeSession	Ends a webservices session with the switch.
createProfile	Creates a new profile.
createTimer	Creates a new timer on the switch.
deleteProfile	Deletes an existing profile.
deleteTimer	Deletes an existing timer from the switch.
editProfile	Modifies an existing profile.
getAllProfile	Gets all the profiles from the switch.
getAllTimers	Gets all the timers from the switch.
getExecutionResult	Gets an execution result.

Table 3: UPM APIs Provided in the UPM.WSDL File (Continued)

Operation	Description
getProfile	Gets the details of a existing profile.
keepAlive	Keeps the webservices session open during periods of inactivity.
openSession	Starts a webservices session with the switch.
runScript	Runs a script when an event occurs on the switch.
saveConfig	Saves the switch configuration from memory to file.
unbindProfile	Removes the binding of a profile from ports.

ExtremeXOS APIs

The available ExtremeXOS APIs are shown in Table 4. The APIs handle VLANs, FDBs, dynamic ACLs, policies, configurations, and CLI commands.

Table 4: ExtremeXOS APIs Provided in the XOS.WSDL File

Operation	Description
createVlan	Creates a VLAN.
deleteVlan	Deletes a VLAN.
getVlan	Gets a VLAN.
getAllVlan	Gets all VLANs.
createFdb	Creates an FDB.
deleteFdb	Deletes an FDB.
getFdb	Gets an FDB.
getAllFdb	Gets all FDBs.
savePolicy	Saves the named policy file onto an ExtremeXOS switch.
deletePolicy	Removes the named policy file from an ExtremeXOS switch.
getPolicy	Retrieves the named policy file from an ExtremeXOS switch.
refreshPolicy	Refreshes the policy on all interfaces to which it is applied.
listPolicy	Lists the names of policy files on the ExtremeXOS switch.
bindPolicyOnInterface	Applies the policy to an interface. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
unbindPolicyOnInterface	If a policy name is specified, it removes the policy from all interfaces to which it is applied. If the policy name is NOT specified, it removes any policy that is currently applied on the interface specified. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
listPolicyOnInterface	Lists the name of the policy applied on the specified interface. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
setDynamicAcl	Creates or overwrites a dynamic ACL rule on an ExtremeXOS switch.
deleteDynamicAcl	Deletes a dynamic ACL rule from an ExtremeXOS switch.
getDynamicAcl	Retrieves the named dynamic ACL rule from an ExtremeXOS switch.
listDynamicAcl	Lists the names of dynamic ACLs on the ExtremeXOS switch.

Table 4: ExtremeXOS APIs Provided in the XOS.WSDL File (Continued)

Operation	Description
insertDynamicAclOnInterface	Inserts the dynamic ACL on the specified interface before or after an existing dynamic ACL rule. If no existing dynamic ACL rule is specified, the new dynamic ACL is inserted either as the first or the last ACL rule. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
removeDynamicAclOnInterface	Removes the dynamic ACL from the specified interface. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
removeDynamicAclOnAllInterfaces	Removes the dynamic ACL from all interfaces to which it is bound.
listDynamicAclOnInterface	Lists the name of dynamic ACLs applied on the specified interface. An interface can be either a VLAN, a port, or the special ANY interface. If neither VLAN nor port interface is specified, it refers to the ANY interface.
execConfig	Executes the ExtremeXOS native XML configuration.
execCli	Executes an ExtremeXOS CLI command. The output of the command is returned. Input to the operation is a string which takes in the command you want to execute. Output is a string which is the dump fo the screen display. Caveats/assumptions are detailed in the API reference.

Programming with Function-Based APIs (xos.wsdl)

1 Use basic HTTP authentication.

```
XosLocator xl=new XosLocator();
java.net.URL url=new java.net.URL("http://
"+device+"/xmlService");

XosPortType stub=xl.getXosPort(url);

((org.apache.axis.client.Stub)stub).setUsername
(username);
((org.apache.axis.client.Stub)stub).setPassword
(password);
```

User name and password for authentication

2 Send the API request.

This example calls insertDynamicAclOnInterface which applies a dynamic ACL, myAcl, to port 1:3.

```
stub.insertDynamicAclOnInterface("", “1:3”, “myAcl”, "", AclInsertDirection.AFTER);
```

Debugging

The XML server’s logging mechanism provides tracing for debugging problems. Use the following sequence to capture the log.

1 Enable tracing using the following CLI command:

```
debug xmld enable tracing
```

The XMLD process logs all subsequent request and response data into a log file named Trace.xmld.<pid>, located in the internal scratch partition.

2 Disable tracing using the following CLI command:

```
debug xmld disable tracing
```

Any tracing related to XMLD is deleted at start up.

Related Publications

The publications related to this one are:

- *ExtremeXOS Concepts Guide*
- *ExtremeXOS Command Reference Guide*

Check for the latest versions of documentation on the Extreme Networks documentation website at:

http://www.extremenetworks.com/go/documentation