Cisco CallManager 4.1(3) AXL Serviceability API Programming Guide

This document describes the implementation of AXL-Serviceability APIs that are based on version 3.3.0.1 or higher. Cisco CallManager Real-Time information, Performance Counters, and Database information exposure occurs through an AXL-Serviceability Interface that confirms to the World Wide Web Consortium (W3C) standard. The Web Service Description Language (WSDL) files provide interface definitions.

To access all AXL API downloads and AXL requests and responses found in this document, refer to the following URL:

http://www.cisco.com/pcgi-bin/dev_support/access_level/product_support

Contents

This document covers the following topics:

Corporate Headquarters:
Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

Introduction

By exposing Cisco CallManager real-time information, performance counter, and database information, customers can write customized applications. AXL-Serviceability APIs, an extensible SOAP-based XML web service, conforms to the SOAP Spec 1.1 [1] and the WSDL Spec 1.1 [2]. AXL-Serviceability APIs represent one server component of the Cisco Serviceability product.

AXL-Serviceability APIs provides RPC-style operations for the clients. Clients of AXL-Serviceability APIs can run in different OS platforms and can communicate with AXL-Serviceability APIs through the standard SOAP protocol. The AXL-Serviceability APIs includes a goal to provide access to the core Cisco Serviceability functionalities through an open and standard transport protocol and data model.

The following list gives the Serviceability services that are exposed via AXL-Serviceability APIs:

- Windows 2000 Perfmon Data Collection.
- Cisco CallManager Device RIS Search

The AXL-Serviceability APIs gets implemented as an ISAPI component of the IIS. AXL-Serviceability APIs as an extensible service allows external implementation of SOAP ports that are to be plugged in through the SOAP port extension dll. This document describes the implementation of AXL-Serviceability APIs that are based on version 3.3.0.1 or higher. The latest implementation of AXL-Serviceability APIs supports only the Perfmon Data Collection functionality and Real-Time information. Cisco plans to add more Serviceability functionalities to the AXL-Serviceability APIs in the near future.

Data Model

AXL-Serviceability APIs are based on SOAP with XML request and response messages. They must conform to the structure of a SOAP Envelope element. Although SOAP is a standard protocol, many of its data model aspects are left open for flexibility reasons. For example, it permits different transport protocols such as SMTP, FTP, or HTTP to carry SOAP messages. The implementation of a SOAP service must specify the bindings of the data model to constitute a concrete wire protocol specification.

WSDL 1.1[2] provides the mechanism to describe the complete SOAP bindings that AXL-Serviceability APIs use.

The following Cisco CallManager server file contains perfmon SOAP requests service definitions:

http://ASTSERVERNAME/soap/AstService.wsdl

The following file contains Cisco CallManager and CTI manager SOAP real-time information requests definitions:

http://ASTSERVERNAME/soap/risport.wsdl

To know what services are available, how to form the request message, and to interpret the response message properly, you must download these files. Basically, the WSDL file has everything that you need to know about AXL-Serviceability APIs. You can get the WSDL file as a static XML page.

ASTSERVERNAME refers to the server on which the AXL-Serviceability APIs is running.

Soap Binding

The binding section of the Serviceability SOAP WSDL files well specifies AXL-Serviceability APIs SOAP binding information. Binding specifications apply to both request and response messages. SOAP Binding covers the following aspects.

Character Encoding

AXL-Serviceability APIs use UTF-8 to encode the data stream in both request and response SOAP messages. The encoding attribute of the XML declaration specifies UTF-8 encoding.
AXL-Serviceability APIs also sets “text/xml; charset='utf-8'” as the value of the Content-Type response header field. Internally, AXL-Serviceability APIs processes the data by using UCS-2 Unicode code page.

Binding Style

AXL-Serviceability APIs uses RPC binding style. In SOAP, the word operation refers to method or function. Each AXL-Serviceability APIs operation call gets modeled as a remote procedure call that is encapsulated in SOAP messages. The HTTP request carries the RPC calls while the HTTP response carries the call returns. The call information is modeled as a structure. The member elements of the body entry represent the accessor elements which represent the input parameters. Similar to the request, the response data structure is also modeled as a structure with accessors that correspond to output parameters. Parameters that are both input and output must appear in both the request and response message.

Transport

SOAP allows different transport protocols to carry SOAP messages. AXL-Serviceability APIs use the standard HTTP as its transport. The clients use the POST verb to send request to AXL-Serviceability APIs.

Note AXL-Serviceability APIs do not use the M-POST method as defined in the HTTP Extension Framework.

Encoding Rule

AXL-Serviceability APIs follow the recommended data model and serialization/encoding rules as defined in Section 5.1 of SOAP Spec 1.1[1] for both the request and response messages. SOAP simple types are based on the built-in data types that are defined in XML Schema Part 2[4]. AXL-Serviceability APIs define its own data types, which are derived from the built-in types. The schemas element of the AXL-Serviceability APIs WSDL file specifies AXL-Serviceability APIs that are derived data types. AXL-Serviceability APIs use both simple and compound data types, such as arrays and structures. All operations in AXL-Serviceability APIs pass parameters by value.

For performance reasons, AXL-Serviceability APIs do not specify the size of the array elements in the response message. It leaves the size as [], which means that no particular size is specified, but the clients can determine the size by enumerating the actual number of elements that are inside the array. Point 8 of section 5.1 of SOAP Spec 1.1 [1] specifies this.

The following target namespace URL for AXL-Serviceability APIs data types schema applies:
xmlns:axl =http://www.cisco.com/AXL/API/1.0

AXL-Serviceability APIs qualifies the first body entry in the response, which represents the call-return, with this target namespace. Similarly, clients of AXL-Serviceability APIs also need to qualify the first body entry in the request, which represents the call, with this namespace.

Request

This section describes the SOAP protocol specifications that are relevant only to the request message. With RPC-style SOAP binding, the request SOAP message contains the operation call information that is encoded as a struct. The call struct appears as the first body entry in the request message. The call struct basically contains the name of the operation and the input parameters. The name of the top-level element gets set to be the operation name. The struct contains accessor element members that represent the input parameters. Operation that takes no parameters will have no members in the struct. The names of the accessor elements get set to be the same as the names of the input parameters. The values of the accessor elements represent the values of the input parameters. The order of the accessor elements must match the order of the input parameters as specified in the signature of the operation.

SOAP Action Header

AXL-Serviceability APIs requires SOAP clients to include the SOAP Action HTTP header field in the request message. The *Simple Object Access Protocol (SOAP) 1.1* specification does not place any restrictions on the format of this header. For AXL-Serviceability APIs, the `soapAction` attribute of the SOAP element, which is defined under the binding section of the Serviceability SOAP API WSDL files, specifies the format of the SOAP Action header. All AXL-Serviceability APIs operations use the following URI format:

```
http://schemas.cisco.com/ast/soap/action/#PortName#OperationName"
```

Note Because enclosing double-quote characters are part of the URI, the header must include them.

PortName acts as a placeholder that refers to the name of the port. AXL-Serviceability APIs must support the port. OperationName represents a placeholder that refers to the name of the intended operation within the specified port. This name must match the operation name in the request body, which is specified as the name of the first body entry element. The SOAPAction header gets used to indicate the intent of the SOAP request without having to parse the body of the request. A SOAP server can check the value of this header first and determine whether to fail the operation before it proceeds with the XML parsing of the body. This gives some efficiency edge on the server side. This scheme also allows non-SOAP-aware intermediary HTTP servers or proxies to determine the intent of the payload.

Port

A SoapPort basically represents an instantiation of a SoapBinding with a specific network address. Because AXL-Serviceability APIs use HTTP as the transport protocol, the network address in this case specifies an HTTP request URL. SoapPortType, with specific binding rules added to it, provide a basis for the definition of SoapBinding. The service section of the WSDL file defines the request URL for all AXL-Serviceability APIs operations as follows:

```
http://ASTSERVERNAME/soap/astsvc.dll
```

ASTSERVERNAME refers to the server name on which the Serviceability SOAP service is running. Every request for the AXL-Serviceability APIs operations must use this URL. The word SOAP refers to the virtual root on which “astsvc.dll” gets placed.

SOAP Header

As previously explained, the SOAP header provides a general way to add features to SOAP messages in a decentralized fashion with no prior contract between the sender and recipient. AXL-Serviceability APIs do not use this feature, so no `Header` element is expected in the envelope and gets ignored. If the `Header` element gets included with the `mustUnderstand` attribute set to 1, AXL-Serviceability APIs reply with a fault and fault code value that is set to `MustUnderstand`.

The following example shows AXL-Serviceability APIs SOAP request message with the HTTP header information:

Example

```
POST /soap/astsvc.dll HTTP/1.1
Host: nozomi
Content-Type: text/xml; charset=utf-8
Content-Length: xxx
SOAPAction: "http://schemas.cisco.com/ast/soap/action/#PerfmonPort#PerfmonOpenSession"

<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <q1:PerfmonOpenSession xmlns:q1="http://schemas.cisco.com/ast/soap/" />
        </soap:Body>
</soap:Envelope>
```

Response

This section describes the SOAP protocol specifications that are relevant only to the response message.

For a successful operation call, the call-return gets encoded as a structure. The structure appears as the first body entry of the response. The call-return basically contains the output parameters or return values of the call. The name of the structure top-level element has no significance, and *Simple Object Access Protocol (SOAP) 1.1* specification does not place any restriction on the name. The structure contains accessor member elements, which represent the output parameters of the call. The names of the accessor elements are the same as the names of the output parameters. The contents of the accessor elements represent the values of the output parameters. The order of the accessor elements must match the order of output parameters as specified in the operation signatures. Operation, which does not return any value, contain no member elements in the call-return struct. AXL-Serviceability APIs use the following naming conventions for its call-return top-level element:

```
OperationNameResponse
```

OperationName represents a placeholder that refers to the name of the operation that is specified in the corresponding request message. This format is specific only for AXL-Serviceability APIs implementation.

The target namespace that is described in the Encoding Rule section, qualifies the call-return. AXL-Serviceability APIs returns HTTP status 200 when the operation succeeds.

For a failed operation call, AXL-Serviceability APIs includes the SOAP `fault` element in the response body. Similar to call-return, the `fault` element also appears as the first body entry. AXL-Serviceability APIs sets HTTP 500 status when sending fault message.

Fault

When AXL-Serviceability APIs process a request and detect that an error occurred, it replies with a SOAP `fault` element in the response. The `fault` element appears as the first response body entry. The `fault` element comprises the following four subelements:

- Fault Code
- FaultString
- FaultActor
- Detail

Fault Code

AXL-Serviceability APIs uses the following standard fault code values as defined in Section 4.4.1 of the *Simple Object Access Protocol (SOAP) 1.1* specification.

VersionMismatch

This fault code gets set when the namespace URL of the request envelope does not match xmlns:axl =http://www.cisco.com/AXL/API/1.0.

MustUnderstand

This fault code gets set when the clients include the `Header` element in the envelope along with the `mustUnderstand` attribute set to 1. AXL-Serviceability APIs do not use the `Header` element. See the SOAP Header section for details.

Client

This fault code gets set when AXL-Serviceability APIs encounters errors that are related to the clients.

Server

This fault code gets set when AXL-Serviceability APIs encounter errors that are related to the service itself.

The `faultcode` value gets qualified with xmlns:axl =http://www.cisco.com/AXL/API/1.0 namespace. This subelement always exists in the `fault` element as the *Simple Object Access Protocol (SOAP) 1.1* specification specifies. This represents a general classification of errors.

FaultString

AXL-Serviceability APIs set a short error description in the `faultstring` element that is intended for human reading. Similar to `faultcode`, this subelement remains always present as required by the *Simple Object Access Protocol (SOAP) 1.1* specification requires. The string value of the FaultString is specific only to the AXL-Serviceability APIs.

FaultActor

AXL-Serviceability APIs does not set this subelement. Note that a proxy or intermediary server must include this subelement if it generates a fault message.

Detail

If AXL-Serviceability APIs parse and process the request body and determine that an error occurs afterwards, it includes the detailed error information in the `detail` subelement.

If AXL-Serviceability APIs do not include the `detail` subelement in the fault element, the error does not relate to the request body. Data types that are defined in the AXL-Serviceability APIs. WSDL files specify the encoding rule for the `detail` element. The following fragments of the file describe the types:

```
 64<complexType name='CallInfoType'>
 65  <sequence>
 66      <element name='FileName' type='xsd:string'/>
 67      <element name='LineNo' type='xsd:int'/>
 68      <element name='ErrorCode' type='xsd:unsignedInt'/>
 69      <element name='Function' type='xsd:string'/>
 70      <element name='Params' type='xsd:string'/>
 71  </sequence>
 72</complexType>
 73
 74<complexType name='ErrorInfoType'>
 75  <sequence>
 76      <element name='Version' type='xsd:string'/>
 77      <element name='Time' type='xsd:time'/>
 78      <element name='ProcId' type='xsd:unsignedInt'/>
 79      <element name='ThreadId' type='xsd:unsignedInt'/>
 80      <element name='ArrayOfCallInfo' type='tns:ArrayOfCallInfoType'/>
 81  /sequence>
82  </complexType>

128<complexType name='ArrayOfCallInfoType'>
129  <complexContent>
130      <restriction base='SOAP-ENC:Array'>
131          <sequence>
132              <element name='CallInfo'
133                  type='tns:CallInfoType' minOccurs='1' maxOccurs='unbounded'/>
134          </sequence>
135      </restriction>
136  </complexContent>
137</complexType>
```

AXL-Serviceability APIs name the `detail` entry element as `ErrorInfo` and the type as `ErrorInfoType`. This type provides a structure with several accessor elements. The `Version` accessor contains the build version of the astsvc.dll. The `Time` accessor denotes the time when the error occurs. The `ProcId` accessor contains the process ID of the AXL-Serviceability APIs, and, in this case, it should be the same process as the inetinfo.exe (IIS process). The `ThreadId` accessor contains the thread ID that generates the fault. The `ArrayOfCallInfo` accessor contains an array of `CallInfo` elements.

The type for `CallInfo` specifies `CallInfoType` and also represents a structure. `CallInfoType` contains detailed information that describes where the error occurs in the codes. It also includes the returned error code of the function, and the parameter data. The `CallInfoType` design allows capturing as much information as needed, so it is easy and fast to track down and investigate a problem. Depending on the implementation of the operation, several `CallInfo` elements can exist in the array.

The following example shows a successful AXL-Serviceability APIs SOAP response message with HTTP headers:

Example

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: xxx
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonOpenSessionResponse xmlns:m="http://schemas.cisco.com/ast/soap/">
            <SessionHandle>{01944B7E-183F-44C5-977A-F31E3AE59C4C}</SessionHandle>
        </m:PerfmonOpenSessionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following example shows a failed AXL-Serviceability APIs SOAP response message with HTTP headers:

Example

```
HTTP/1.1 500 OK
Content-Type: text/xml; charset=utf-8
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>Perfmon error occurs</faultstring>
            <detail>
                <m:ErrorInfo xmlns:m="http://schemas.cisco.com/ast/soap/">
                    <Version>3.2.0.2</Version>
                    <Time>07/16/2001 - 00:00:24</Time>
                    <ProcId>1200</ProcId>
                    <ThreadId>300</ThreadId>
                    <ArrayOfCallInfo SOAP-ENC:arrayType="m:CallInfoType[]">
                        <CallInfo>
                            <FileName>perfmon.cpp</FileName>
                            <LineNo>396</LineNo>
                            <ErrorCode>3221228473</ErrorCode>
                            <Function>AddCounter</Function>
                            <Params>\\nozomi\tcp\Bad Counter Name</Params>
                        </CallInfo>
                    </ArrayOfCallInfo>
                </m:ErrorInfo>
            </detail>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Perfmon error codes can be refered form Microsoft site.  Above error 3221228473 code refer
to
0xC0000BB9 The specified counter could not be found.PDH_CSTATUS_NO_COUNTER

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/pdh_error_co
des.asp
```

Namespaces

AXL-Serviceability APIs uses the following XML namespaces:

- xmlns:axl =http://www.cisco.com/AXL/API/1.0

  This represents the namespace URI for SOAP envelope.

- xmlns:axl =http://www.cisco.com/AXL/API/1.0

  This represents the namespace for the SOAP-recommended encoding rule that is based on XML Schema.

- xmlns:axl =http://www.cisco.com/AXL/API/1.0

  This represents the namespace URL for AXL-Serviceability APIs data types as defined in the WSDL file.

Highly Isolated IIS Service

In this release, SOAP moves to highly isolated process that does not run as part of IIS thread. Faults on SOAP service, therefore, will not affect IIS process.

Components and Dependency

AXL-Serviceability APIs comprises the following module:

- Astsvc.dll provides the base implementation of SOAP Serviceability APIs.
- AstService.wsdl and RisPort.wsdl represents XML files that describe the web service.
- Real-time information server (Cisco RIS Data Collector).

AXL-Serviceability APIs Ports

This section explains in full details the interface of SOAP service built-in ports.

PerfmonPort

PerfmonPort comprises several operations that allow clients to do the following perfmon-related tasks:

- Collect perfmon counter data.

  AXL-Serviceability APIs provides two ways to collect perfmon data: session-based and single-transaction.

- Get the list of all perfmon object and counters names that are installed in a particular host.
- Get the list of current instances of a perfmon object.
- Get textual description of a perfmon counter.

The 'perfport.dll' SOAP extension implements PerfmonPort. For a short overview of Perfmon Monitoring in Windows 2000, refer to following links:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/performance_objects_and_counters.asp

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/performance_data.asp

PerfmonListCounter

This operation returns the list of Perfmon objects and counters in a particular host.

Request Format

The PerfmonListCounter operation takes a single parameter:

Parameter	Type	Derived From	Description
Host	`xsd:string`	`xsd:string`	Contains the name or address of the target server from which to get the counter information.

The following example shows a PerfmonListCounter request:

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonListCounter xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <Host xsi:type="xsd:string">nozomi</Host>
        </q1:PerfmonListCounter>
    </soap:Body>
</soap:Envelope>
```

Response Format

PerfmonListCounter returns with information that describes the hierarchical structures of the Perfmon objects and counters. The body entry includes an `ArrayOfObjectInfo` element. The following fragments from the AXL-Serviceability APIs .WSDL file describe the types that this response uses:

```
   106  <complexType name='ArrayOfObjectInfoType'>
   107      <complexContent>
   106          <restriction base='SOAP-ENC:Array'>
   109              <sequence>
   110                  <element name='ObjectInfo'
   111                      type='tns:ObjectInfoType' minOccurs='1' maxOccurs='unbounded'/>
   112              </sequence>
   113          </restriction>
   114      </complexContent>
115 </complexType>
```

The `ArrayOfObjectInfo` element comprises an array of `ObjectInfo` elements that have the following type:

```
    56  <complexType name='ObjectInfoType'>
    57      <sequence>
    58          <element name='Name' type='tns:ObjectNameType'/>
    59          <element name='MultiInstance' type='xsd:boolean'/>
    60          <element name='ArrayOfCounter' type='tns:ArrayOfCounterType'/>
    61      </sequence>
62  </complexType>


    24  <simpleType name='ObjectNameType'>
    25      <restriction base='string'/>
    26  </simpleType>
```

The `Name` element, whose type is derived from string, describes the name of the object. `MultiInstance` element indicates whether the object has more than one instance. The `ArrayOfCounter` element acts as a container for an array of `Counter` elements that have the following types:

```
    44  <complexType name='CounterType'>
    45      <sequence>
    46          <element name='Name' type='tns:CounterNameType'/>
    47      </sequence>
48  </complexType>
    32  <simpleType name='CounterNameType'>
    33      <restriction base='string'/>
    34  </simpleType>
```

The `Name` element, whose type is derived from `xsd:string`, describes the name of the counter.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonListCounterResponse xmlns:m="http://schemas.cisco.com/ast/soap/">
            <ArrayOfObjectInfo SOAP-ENC:arrayType="m:ObjectInfoType[]">
                <ObjectInfo>
                    <Name>.NET CLR Memory</Name>
                    <MultiInstance>true</MultiInstance>
                    <ArrayOfCounter SOAP-ENC:arrayType="m:CounterType[]">
                        <Counter>
                            <Name># Gen 0 Collections</Name>
                        </Counter>
                        <Counter>
                            <Name># Gen 1 Collections</Name>
                        </Counter>
                        ...
                    </ArrayOfCounter>
                </ObjectInfo>
                <ObjectInfo>
                    <Name>.NET CLR LocksAndThreads</Name>
                    <MultiInstance>true</MultiInstance>
                    <ArrayOfCounter SOAP-ENC:arrayType="m:CounterType[]">
                        <Counter>
                            <Name>Total # of Contentions</Name>
                        </Counter>
                        <Counter>
```

```
                        <Name>Contention Rate / sec</Name>
                    </Counter>
                    <Counter>
                        <Name>Current Queue Length</Name>
                    </Counter>
                    ...
                </ArrayOfCounter>
            </ObjectInfo>
            ...
        </ArrayOfObjectInfo>
    </m:PerfmonListCounterResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonListInstance

This operation returns the list of instances of a perfmon object in a particular host. Instances of an object can dynamically come and go, and this operation returns the most recent list of instances.

Request Format

The PerfmonListIntance operation takes the following parameters:

Parameter	Type	Derived From	Description
Host	`xsd:string`	`xsd:string`	Contains the address of the target server on which the object resides.
Object	`xsd:string`	`xsd:string`	Contains the name of the object.

The following example shows the `PerfmonListInstance` request with nozomi host and `Process` object parameters.

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonListInstance xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <Host xsi:type="xsd:string">nozomi</Host>
            <Object xsi:type="xsd:string">Process</Object>
        </q1:PerfmonListInstance>
    </soap:Body>
</soap:Envelope>
```

Response Format

`PerfmonListInstance` returns an element named `ArrayOfInstance`. The type for this element specifies `ArrayOfInstanceType`, which is an array of `Instance` elements. The following fragments from AXL-Serviceability APIs .WSDL file explain the types that this response uses:

```
    84<complexType name='ArrayOfInstanceType'>
    85  <complexContent>
    86      <restriction base='SOAP-ENC:Array'>
    87          <sequence>
    88              <element name='Instance'
    89                  type='tns:InstanceType' minOccurs='0' maxOccurs='unbounded'/>
    90          </sequence>
    91      </restriction>
    92  </complexContent>
93  </complexType>
```

Note `ArrayOfInstanceType` can have 0 (zero) `Instance` element, in which case the requested object is not of a `multi-instance` object.

```
    50<complexType name='InstanceType'>
    51  <sequence>
    52      <element name='Name' type='tns:InstanceNameType'/>
    53  </sequence>
54  </complexType>
```

The type for `Instance` element specifies `InstanceType`. It represents a structure with a single-element member: `Name`.

```
    28<simpleType name='InstanceNameType'>
    29  <restriction base='string'/>
    30</simpleType>
```

The `Name` element, whose type is `InstanceNameType`, which is derived from `xsd:string`, contains the name of the instance of the requested object.

The following example shows the response to the request in the previous example:

Example

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonListInstanceResponse xmlns:m="http://schemas.cisco.com/ast/soap/">
            <ArrayOfInstance SOAP-ENC:arrayType="m:InstanceType[]">
                <Instance>
                    <Name>Idle</Name>
                </Instance>
                <Instance>
                    <Name>System</Name>
                </Instance>
                <Instance>
                    <Name>SMSS</Name>
                </Instance>
                <Instance>
```

```
                    <Name>CSRSS</Name>
                </Instance>
                <Instance>
                    <Name>WINLOGON</Name>
                </Instance>
                <Instance>
                    <Name>SERVICES</Name>
                </Instance>
                <Instance>
                    <Name>_Total</Name>
                </Instance>
                ...
            </ArrayOfInstance>
        </m:PerfmonListInstanceResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonQueryCounterDescription

This operation returns the help text of a particular counter.

Request Format

PerfmonQueryCounterDescription takes the following parameter:

Parameter	Type	Derived From	Description
Counter	`CounterNameType`	`xsd:string`	Contains the name of the counter.

The following example shows the `PerfmonQueryCounterDescription` request:

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonQueryCounterDescription
            xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <Counter xsi:type="xsd:string">\\nozomi\Server\Files Open</Counter>
        </q1:PerfmonQueryCounterDescription>
    </soap:Body>
</soap:Envelope>
```

Response Format

PerfmonQueryCounterDescription returns an element named `HelpText` that is of the `xsd:string` type. It contains the help text of the requested counter.

The following example shows the response to the request in the previous example.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonQueryCounterDescriptionResponse
            xmlns:m="http://schemas.cisco.com/ast/soap/">
            <HelpText>The number of files currently opened in the server. Indicates
current server
activity.
</HelpText>
        </m:PerfmonQueryCounterDescriptionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonOpenSession

Client program submits this operation to get a session handle from the AXL-Serviceability APIs. The client needs a session handle to do the session-based perfmon counter data collection. The session handle represents a universally unique identifier that gets used only once. This guarantees that no duplicate handles exist. AXL-Serviceability APIs keep the opened handles in the cache. If no activity occurs on a handle for the last 25 hours, AXL-Serviceability APIs remove the handle and render it invalid.

In a session-based perfmon data collection, use the following related operations:

- PerfmonOpenSession
- PerfmonAddCounter
- PerfmonRemoveCounter
- PerfmonCollectSessionData
- PerfmonCloseSession

After a client gets a session handle, it normally proceeds to submit the `PerfmonAddCounter` operation call and then follows with the `PerfmonCollectSessionData` operation. `PerfmonCollectSessionData` specifies the main operation that does the collection of the perfmon data for the clients. When the client no longer needs the session handle, it should submit `PerfmonCloseSession,` so the AXL-Serviceability APIs can remove the handle from cache. Clients can dynamically add new counters to the session handle and remove some from it while the session handle is still open. The `PerfmonRemoveCounter` operation does counter removal.

Request Format

The `PerfmonOpenSession` operation takes no parameter.

The following example shows the `PerfmonOpenSession` request:

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <q1:PerfmonOpenSession xmlns:q1="http://schemas.cisco.com/ast/soap/" />
        </soap:Body>
</soap:Envelope>
```

Response Format

`PerfmonOpenSession` returns a single element that is named `SessionHandle`. Its type specifies `SessionHandleType`, which is derived from `xsd:string`, and it contains the guide for the session handle.

The following example shows the `PerfmonOpenSession` response:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonOpenSessionResponse xmlns:m="http://schemas.cisco.com/ast/soap/">
            <SessionHandle>{01944B7E-183F-44C5-977A-F31E3AE59C4C}</SessionHandle>
        </m:PerfmonOpenSessionResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonAddCounter

This operation adds an array of counters to a session handle.

Request Format

PerfmonAddCounter takes the following parameters:

Parameter	Type	Derived From	Description
SessionHandle	`SessionHandleType`	`xsd:string`	Contains the session handle that the PerfmonOpenSession operation previously opened
ArrayOfCounter	`ArrayOfCounterType`	an array of `Counter` elements	Contains the name of the counter to be added to the session handle

The following fragments from AXL-Serviceability APIs describe the types that this request uses:

```
    95<complexType name='ArrayOfCounterType'>
    96  <complexContent>
    97      <restriction base='SOAP-ENC:Array'>
    98          <sequence>
    99              <element name='Counter'
   100                  type='tns:CounterType' minOccurs='1' maxOccurs='unbounded'/>
   101          </sequence>
   102      </restriction>
   103  </complexContent>
104 </complexType>
```

Note `ArrayOfCounterType` expects at least one `Counter` element in the array.

```
    44<complexType name='CounterType'>
    45  <sequence>
    46      <element name='Name' type='tns:CounterNameType'/>
    47  </sequence>
48  </complexType>
```

`CounterType` represents a structure, and it has a single element member: `Name`.

```
    32<simpleType name='CounterNameType'>
    33  <restriction base='string'/>
34  </simpleType>
```

The `Name` element that is of string-derived type contains the name of the counter.

The following example shows the `PerfmonAddCounter` request with two counters. This example uses a single-reference accessor.

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://tempuri.org/"
    xmlns:types="http://tempuri.org/encodedTypes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonAddCounter xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <SessionHandle xsi:type="xsd:string">
                {1A490F1E-D82C-403F-9CF0-C4D4ABD6FF3E}
            </SessionHandle>
            <ArrayOfCounter soapenc:arrayType="q1:CounterType[2]">
                <Counter>
                    <Name>\\nozomi\process(inetinfo)\handle count</Name>
                </Counter>
                <Counter>
                    <Name>\\nozomi\process(csrss)\handle count</Name>
                </Counter>
            </ArrayOfCounter>
</q1:PerfmonAddCounter">
    </soap:Body>
</soap:Envelope
```

The following shows an example of the `PerfmonAddCounter` request with three counters in the ArrayOfCounter parameter.

Note This example uses multi-reference accessors.

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonAddCounter xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <SessionHandle xsi:type="xsd:string">
                {1A490F1E-D82C-403F-9CF0-C4D4ABD6FF3E}
            </SessionHandle>
            <ArrayOfCounter href="#id1"/>
        </q1:PerfmonAddCounter>
        <soapenc:Array id="id1"
            xmlns:q2="http://schemas.cisco.com/ast/soap/"
            soapenc:arrayType="q2:CounterType[3]">
            <Item href="#id2" />
            <Item href="#id3" />
            <Item href="#id4" />
        </soapenc:Array>
        <q3:CounterType id="id2" xsi:type="q3:CounterType"
            xmlns:q3="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Segments/sec</Name>
        </q3:CounterType>
        <q4:CounterType id="id3" xsi:type="q4:CounterType"
            xmlns:q4="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Connections Reset</Name>
        </q4:CounterType>
        <q5:CounterType id="id4" xsi:type="q5:CounterType"
            xmlns:q5="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Connections Active</Name>
        </q5:CounterType>
    </soap:Body>
</soap:Envelope>
```

Response Format

The following example shows that the `PerfmonAddCounter` returns no output:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonAddCounterResponse xmlns:m="http://schemas.cisco.com/ast/soap/"/>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If `PefmonAddCounter` fails to add one or more counters that are specified in the request, AXL-Serviceability APIs reply with a fault response. Some counters that are specified in the request may get successfully added, while others failed to be added.

In this case, AXL-Serviceability APIs reply with a fault. The `Params` element of `CallInfo` element specifies each failed counter name. Client programs can conclude that counter names, which are specified in the request but do not appear in the fault message, actually get added successfully to the query handle.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
            <faultstring>Perfmon error occurs</faultstring>
            <detail>
                <m:ErrorInfo xmlns:m="http://schemas.cisco.com/ast/soap/">
                    <Version>3.2.0.1</Version>
                    <Time>07/15/2001 - 00:00:24</Time>
                    <ProcId>1212</ProcId>
                    <ThreadId>340</ThreadId>
                    <ArrayOfCallInfo SOAP-ENC:arrayType="m:CallInfoType[]">
                        <CallInfo>
                            <FileName>perfmon.cpp</FileName>
                            <LineNo>396</LineNo>
                            <ErrorCode>3221228473</ErrorCode>
                            <Function>AddCounter</Function>
                            <Params>\\nozomi\tcp\Invalid Counter Name 1</Params>
                        </CallInfo>
                        <CallInfo>
                            <FileName>perfmon.cpp</FileName>
                            <LineNo>396</LineNo>
                            <ErrorCode>3221228473</ErrorCode>
                            <Function>AddCounter</Function>
                            <Params>\\nozomi\tcp\Invalid Counter Name 2</Params>
                        </CallInfo>
                    </ArrayOfCallInfo>
                </m:ErrorInfo>
            </detail>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonRemoveCounter

This operation removes an array of counters from a session handle.

Request Format

`PerfmonRemoveCounter` takes the following parameters:

Parameter	Type	Derived From	Description
SessionHandle	`SessionHandleType`	`xsd:string`	Contains the session handle that the PerfmonOpenSession operation previously opened
ArrayOfCounter	`ArrayOfCounterType`	an array of `Counter` elements	Contains the name of the counter to be added to the session handle

The following example shows a `PerfmonRemoveCounter` request with three counters in the `ArrayOfCounter` parameter.

Note This example uses single-reference accessor style.

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://tempuri.org/"
    xmlns:types="http://tempuri.org/encodedTypes"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonRemoveCounter xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <SessionHandle xsi:type="xsd:string">
                {1A490F1E-D82C-403F-9CF0-C4D4ABD6FF3E}
            </SessionHandle>
            <ArrayOfCounter soapenc:arrayType="q1:CounterType[2]">
                <Counter>
                    <Name>\\nozomi\process(inetinfo)\handle count</Name>
                </Counter>
                <Counter>
                    <Name>\\nozomi\process(csrss)\handle count</Name>
                </Counter>
<Counter>
                    <Name>\\nozomi\process(regsvc)\handle count</Name>
                </Counter>

            </ArrayOfCounter>
</q1:PerfmonRemoveCounter">
    </soap:Body>
</soap:Envelope
```

The following example shows a `PerfmonRemoveCounter` that uses multi reference accessors.

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonRemoveCounter xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <SessionHandle xsi:type="xsd:string">
                {1A490F1E-D82C-403F-9CF0-C4D4ABD6FF3E}
            </SessionHandle>
            <ArrayOfCounter href="#id1" />
        </q1:PerfmonRemoveCounter>
        <soapenc:Array id="id1" xmlns:q2="http://schemas.cisco.com/ast/soap/"
            soapenc:arrayType="q2:CounterType[3]">
            <Item href="#id2" />
            <Item href="#id3" />
            <Item href="#id4" />
        </soapenc:Array>
        <q3:CounterType id="id2" xsi:type="q3:CounterType"
            xmlns:q3="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Segments/sec</Name>
        </q3:CounterType>
        <q4:CounterType id="id3" xsi:type="q4:CounterType"
            xmlns:q4="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Connections Reset</Name>
        </q4:CounterType>
        <q5:CounterType id="id4" xsi:type="q5:CounterType"
            xmlns:q5="http://schemas.cisco.com/ast/soap/">
            <Name xsi:type="xsd:string">\\nozomi\tcp\Connections Active</Name>
        </q5:CounterType>
    </soap:Body>
</soap:Envelope>
```

Response Format

`PerfmonRemoveCounter` returns no data in the response as shown by the following example:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
    <m:PerfmonRemoveCounterResponse xmlns:m="http://schemas.cisco.com/ast/soap/"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If `PefmonRemoveCounter` fails to remove one or more counters that the request specifies, AXL-Serviceability APIs reply with a fault response with similar semantics as `PerfmonAddCounter`. Some of the counters that are specified in the request get successfully removed, while some others failed to be removed. In this case, AXL-Serviceability APIs reply with a fault. The `Params` element of `CallInfo` element specifies each failed counter name. Client programs can conclude that counter names, which are specified in the request but do not appear in the fault message, actually get removed successfully from the query handle.

PerfmonCollectSessionData

This operation collects the perfmon data for all counters that have been added to the query handle.

Request Format

`PerfmonCollectSessionData` takes the following parameter:

Parameter	Type	Derived From	Description
SessionHandle	`SessionHandleType`	`xsd:string`	Contains the session handle that the PerfmonOpenSession operation previously opened

The following example shows a `PerfmonCollectSessionData` request:

Example

```
<?xml version="1.0" encoding="utf-8" ?>
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonCollectSessionData xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <SessionHandle xsi:type="xsd:string">
                {FB343D6D-AA6E-4EDB-9CFA-63A5A3ED6405}
            </SessionHandle>
        </q1:PerfmonCollectSessionData>
    </soap:Body>
</soap:Envelope>
```

Response Format

The `PerfmonCollectSessionData` operation returns the `ArrayOfCounterInfo` element that contains the value and status of all counters that were previously added to the session handle. The type for `ArrayOfCounterInfo` element specifies `ArrayOfCounterInfo`Type, which represents an array of `CounterInfo` elements.

The following fragments from AXL-Serviceability APIs .WSDL show the types that this response uses:

```
   117<complexType name='ArrayOfCounterInfoType'>
   118  <complexContent>
   119      <restriction base='SOAP-ENC:Array'>
   120          <sequence>
   121              <element name='CounterInfo'
   122                  type='tns:CounterInfoType' minOccurs='1' maxOccurs='unbounded'/>
   123          </sequence>
   124      </restriction>
   125  </complexContent>
126 </complexType>
```

`ArrayOfCounterInfoType` has one or more `CounterInfo` elements in the array. The `CounterInfo` element includes the following type:

```
    36<complexType name='CounterInfoType'>
    37  <sequence>
    38      <element name='Name' type='tns:CounterNameType'/>
```

```
    39      <element name='Value' type='xsd:long'/>
    40      <element name='CStatus' type='xsd:unsignedInt'/>
    41  </sequence>
42  </complexType>


    32<simpleType name='CounterNameType'>
    33  <restriction base='string'/>
    34</simpleType>
```

`CounterInfoType` specifies a structure with the following three element members.

Element Name	Description
`Name`	A `CounterNameType`, derived from `xsd:string`, contains the name of the counter that was previously added to the session handle.
`Value`	A 64-bit signed integer (`xsd:long`) that contains the value of the counter
`CStatus`	Indicates whether the value of the counter was successfully retrieved. The type specifies a 32-bit unsigned integer (`xsd:unsignedInt`). First check for the value of `CStatus` element before reading the `Value` element. If the value of `CStatus` equals to 0 or 1, the `Value` element contains a good counter value. Otherwise, it indicates a failure in retrieving the counter value; ignore the `Value` element.

The following example shows a `PerfmonCollectSessionData` response:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
<m:PerfmonCollectSessionDataResponse
xmlns:m="http://schemas.cisco.com/ast/soap/">
            <ArrayOfCounterInfo SOAP-ENC:arrayType="m:CounterInfoType[]">
                <CounterInfo>
                    <Name>\\nozomi\tcp\Segments/sec</Name>
                    <Value>0</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
                <CounterInfo>
                    <Name>\\nozomi\tcp\Connections Reset</Name>
                    <Value>6</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
                <CounterInfo>
                    <Name>\\nozomi\tcp\Connections Active</Name>
                    <Value>23</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
            </ArrayOfCounterInfo>
```

```
        </m:PerfmonCollectSessionDataResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonCloseSession

This operation closes the session handle that the PerfmonOpenSession previously retrieved.

Request Format

PerfmonCloseSession takes the following parameter:

Parameter	Type	Derived From	Description
SessionHandle	`SessionHandleType`	`xsd:string`	Contains the session handle that the `PerfmonOpenSession` operation previously opened

The following example shows a `PerfmonCloseSession` request:

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonCloseSession xmlns:q1="http://schemas.cisco.com/ast/soap/">
<SessionHandle xsi:type="xsd:string">
{01944B7E-183F-44C5-977A-F31E3AE59C4C}
</SessionHandle>
        </q1:PerfmonCloseSession>
    </soap:Body>
</soap:Envelope>
```

Response Format

The following example shows that the `PerfmonCloseSession` does not return data in the response:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonCloseSessionResponse xmlns:m="http://schemas.cisco.com/ast/soap/"/>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

PerfmonCollectCounterData

This operation returns the perfmon data for all counters that belong to an object in a particular host. Unlike the session-based perfmon data collection, this operation collects all data in a single request/response transaction. If the object represents multiple-instance object, this operation always returns the most current instances of the object.

Request Format

PerfmonCollectCounterData takes the following parameters:

Parameter	Type	Derived From	Description
Host	`xsd:string`	`xsd:string`	Contains the address of the target server from which the client wants to get the counter information
Object	`ObjectNameType`	`xsd:string`	Contains the name of the perfmon object

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <q1:PerfmonCollectCounterData xmlns:q1="http://schemas.cisco.com/ast/soap/">
            <Host xsi:type="xsd:string">nozomi</Host>
            <Object xsi:type="xsd:string">Processor</Object>
        </q1:PerfmonCollectCounterData>
    </soap:Body>
</soap:Envelope>
```

Response Format

`PerfmonCollectCounterData` returns an `ArrayOfCounterInfo` element, which is an array of `CounterInfo` elements. The data types that this response uses appears similar to the ones that the response of `PerfmonCollectSessionData` as explained in the PerfmonCollectSessionData, “Response Format” section use.

The following example shows a `PerfmonCollectCounterData` response:

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:PerfmonCollectCounterDataResponse
            xmlns:m="http://schemas.cisco.com/ast/soap/">
            <ArrayOfCounterInfo SOAP-ENC:arrayType="m:CounterInfoType[]">
                <CounterInfo>
                    <Name>\\nozomi\Processor(0)\% Processor Time</Name>
                    <Value>99</Value>
```

```
                    <CStatus>0</CStatus>
                </CounterInfo>
                <CounterInfo>
                    <Name>\\nozomi\Processor(0)\% User Time</Name>
                    <Value>0</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
                <CounterInfo>
                    <Name>\\nozomi\Processor(0)\% Privileged Time</Name>
                    <Value>0</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
                <CounterInfo>
                    <Name>\\nozomi\Processor(0)\Interrupts/sec</Name>
                    <Value>525</Value>
                    <CStatus>0</CStatus>
                </CounterInfo>
                ...
            </ArrayOfCounterInfo>
        </m:PerfmonCollectCounterDataResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Real-Time Information (RisPort)

Selecting Cisco CallManager Real-Time Information

Request Format

SOAP Action and Envelope information

This operation allows clients to perform Cisco CallManager device-related queries. HTTP header should have following SOAP action for these queries.

```
SOAPAction: "http://schemas.cisco.com/ast/soap/action/#RisPort#SelectCmDevices"
```

Query information includes an Envelope as follows:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
3.  xmlns:tns="http://schemas.cisco.com/ast/soap/"
4.  xmlns:types="http://schemas.cisco.com/ast/soap/encodedTypes"
5.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6.  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Session ID

This SOAP header will have session ID that is a unique session ID from client.

```
7.  <soap:Header>
8.  <tns:AstHeader id="id1">
9.  <SessionId xsi:type="xsd:string">SessionId</SessionId>
10. </tns:AstHeader>
11. </soap:Header>
```

Selection Criteria

Selection criteria type, which is a Cisco CallManager Selection criteria, follows the SOAP header.

```
12. <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
13. <tns:SelectCmDevice>
```

If the same information is queried over and over again, send `Stateinfo` from the previous request for each repetitive query by client.

```
14. <StateInfo xsi:type="xsd:string" />
15. <CmSelectionCriteria href="#id1"/>
16. </tns:SelectCmDevice><tns:CmSelectionCriteria id="id1"
xsi:type="tns:CmSelectionCriteria">
```

Maximum Devices Specification

The following example specifies how many maximum devices can be returned for search criteria.

```
17. <MaxReturnedDevices xsi:type="xsd:unsignedInt">10</MaxReturnedDevices>
```

Search Device Classes

The following example specifies the device class type to query for real-time status. Device classes include 'Any', 'Phone', 'Gateway', 'H323', 'Cti', 'VoiceMail', 'MediaResources', 'HuntList', 'SIPTrunk', and 'unknown'.

```
18. <Class xsi:type="tns:DeviceClass">Any</Class>
```

The following example specifies the Model of the device—255 specifies all models.

```
19. <Model xsi:type="xsd:unsignedInt">255</Model>
```

```
MODEL_30SPP = 1,                                                    // Cisco 30 SP+
  MODEL_12SPP = 2,                                                    // Cisco 12 SP+
  MODEL_12SP = 3,                                                     // Cisco 12 SP
  MODEL_12S = 4,                                                      // Cisco 12 S
  MODEL_30VIP = 5,                                                    // Cisco 30 VIP
  MODEL_TELECASTER_BID = 6,                                           // Cisco 7910
  MODEL_TELECASTER_MGR = 7,                                           // Cisco 7960
  MODEL_TELECASTER_BUSINESS = 8,                                      // Cisco 7940
  MODEL_IP_CONFERENCE_PHONE = 9,                                      // Cisco 7935
  MODEL_VGC_PHONE = 10,                                               // Cisco VGC Phone
  MODEL_VGC_BOX_PHONE = 11,                                           // Cisco VGC
Virtual Phone
  MODEL_ATA_186 = 12,                                                 // Cisco ATA 186
  MODEL_SCCP_PHONE = 20,                                              // SCCP Phone
  MODEL_VEGA = 30,                                                    // Analog Access
  MODEL_TITAN1 = 40,                                                  // Digital Access
  MODEL_TITAN2 = 42,                                                  // Digital Access+
  MODEL_LENNON = 43,                                                  // Digital Access
WS-X6608
  MODEL_ELVIS = 47,                                                   // Analog Access
```

```
WS-X6624
  MODEL_CHALICE = 48,                                                    // VGC Gateway
  MODEL_CONF_BRIDGE = 50,                                                // Conference
Bridge
  MODEL_YOKO_CONF_BRIDGE = 51,                                           // Conference
Bridge WS-X6608
  MODEL_DIXIELAND_CFB = 52,                                              // Cisco IOS
Conference Bridge (HDV2)
  MODEL_SUMMIT_CFB = 53,                                                // Cisco Conference
Bridge (WS-SVC-CMM)
  MODEL_H323_PHONE = 61,                                                 // H.323 Phone
  MODEL_H323_GATEWAY = 62,                                               // H.323 Gateway
  MODEL_MUSIC_ON_HOLD = 70,                                              // Music On Hold
  MODEL_DEVICE_PILOT = 71,                                               // Device Pilot
  MODEL_CTI_PORT = 72,                                                   // CTI Port
  MODEL_CTI_ROUTE_POINT = 73,                                            // CTI Route Point
  MODEL_UONE = 80,                                                       // Voice Mail Port
  MODEL_DIXIELAND_SW_MTP = 83,                                           // Cisco IOS
Software Media Termination Point (HDV2)
  MODEL_SUMMIT_APP_CFB = 84,                                             // Cisco Media
Server (WS-SVC-CMM-MS)
  MODEL_FLINT_CONF_BRIDGE  = 85,                                         // Cisco Video
Conference Bridge (IPVC-35xx)
  MODEL_ROUTE_LIST = 90,                                                 // Route List
  MODEL_LOAD_SIMULATOR = 100,                                            // Load Simulator
  MODEL_MTP = 110,                                                       // Media
Termination Point
  MODEL_YOKO_MTP = 111,                                                  // Media
Termination Point Hardware
  MODEL_DIXIELAND_MTP = 112,                                             // Cisco IOS Media
Termination Point (HDV2)
  MODEL_SUMMIT_MTP = 113,                                                // Cisco Media
Termination Point (WS-SVC-CMM)
  MODEL_MGCP_STATION = 120,                                              // MGCP Station
  DEVICE/ ENUM VALUE      DEVICE DESCRIPTION
-------------------------------------------------------------------------------------------------
-----------------
  MODEL_MGCP_TRUNK = 121,                                                // MGCP Trunk
  MODEL_GATEKEEPER = 122,                                                // GateKeeper
  MODEL_14_BUTTON_SIDECAR = 124,                                         // 14-Button Line
Expansion Module
  MODEL_TRUNK = 125,                                                     // Trunk
  MODEL_ANN = 126,                                                       // Tone
Announcement Player
  MODEL_SIP_TRUNK = 131,                                                 // SIP Trunk
  MODEL_SIP_GATEWAY = 132,                                               // SIP Gateway
  MODEL_UNKNOWN_MGCP_GATEWAY = 254,                                      // Unknown MGCP
Gateway
  MODEL_UNKNOWN = 255,                                                   // Unknown
  MODEL_CISCO_IP_PHONE_7905 = 20000,                                     // Cisco 7905
  MODEL_7920 = 30002,                                                    // Cisco 7920
  MODEL_UBIQUITY_I = 30004,                                              // Ubiquity I
  MODEL_IP250D = 30005,                                                  // IP250D
  MODEL_CISCO_7970 = 30006,                                              // Cisco 7970
  MODEL_CISCO_7912 = 30007,                                              // Cisco 7912
  MODEL_CISCO_7902 = 30008,                                              // Cisco 7902
  MODEL_CISCO_SOFTPHONE_SE_M = 30016,                                    // Cisco IP
Communicator
  MODEL_SAMPLEPHONE = 30017,                                             // SamplePhone
  MODEL_CISCO_7965 = 30018,                                              // Cisco 7965
  MODEL_CISCO_7936 = 30019,                                              // Cisco 7936
  MODEL_TANDBERG_VIDEO_PHONE = 30020                                     // Tandberg Video
Phone
```

Device Status in Search Criteria

Specify registered/unregistered status devices. The following example shows status 'Any', 'Registered', 'UnRegistered', 'Rejected', and 'Unknown.'

```
20. <Status xsi:type="tns:CmDevRegStat">Registered</Status>
```

The following example specifies the server name where the search needs to be performed. If no name is specified, a search in all servers in a cluster results.

```
21. <NodeName xsi:type="xsd:string" />

Specify Selection type whether it is IP Address/Name
22. <SelectBy xsi:type="tns:CmSelectBy">Name</SelectBy>
```

Array of Items for Which Search Criteria Are Specified

The following example specifies an array that contains the IP Address or Device Name of the items for which a real-time status is required.

```
23. <SelectItems href="#id2" />Name or IP</tns:CmSelectionCriteria
24. <soapenc:Array id="id2" soapenc:arrayType="tns:SelectItem[2]">
25. <Item href="#id3"/><Item xsi:null="1"/>
26. </soapenc:Array>
27. <tns:SelectItem id="id3" xsi:type="tns:SelectItem">
28. <Item xsi:type="xsd:string"/></tns:SelectItem>
29. </soap:Body>
30. </soap:Envelope>
```

Response Format

The Response follows the following schema and contains information for one to many servers for each server and contains a sequence of search information that is found on the search criteria.

```
<complexType name='SelectCmDeviceResult'>
      <sequence>
            <element name='TotalDevicesFound' type='xsd:unsignedInt'/>
            <element name='CmNodes' type='tns:CmNodes'/>
      </sequence>
</complexType>
```

CMNodes provides a list of CMNodes that are given in search criteria.

```
<complexType name='CmNodes'>
<complexContent>
    <restriction base='SOAP-ENC:Array'>
       <sequence>
    <element name='CmNode' type='tns:CmNode' minOccurs='0 maxOccurs='unbounded'/>
       </sequence>
    </restriction>
</complexContent>
    </complexType>
```

Each CMNode contains a sequence of devices and their registration status.

```
<complexType name='CmNode'>
     <sequence>
        <element name='ReturnCode'          type='tns:RisReturnCode'/>
        <element name='Name'                type='xsd:string'/>
        <element name='NoChange'            type='xsd:boolean'/>
        <element name='CmDevices'           type='tns:CmDevices'/>
      </sequence>
 </complexType>

 <complexType name='CmDevices'>
     <complexContent>
       <restriction base='SOAP-ENC:Array'>
       <sequence>
       <element name='CmDevice' type='tns:CmDevice' minOccurs='0' maxOccurs='200'/>
       </sequence>
       </restriction>
       </complexContent>
     </complexType>
```

The CMDevice information contains the following information.

```
<complexType name='CmDevice'>
<sequence>
<element name='Name' type='xsd:string'/>
<element name='IpAddress' type='xsd:string'/>
<element name='DirNumber' type='xsd:string'/>
<element name='Class' type='tns:DeviceClass'/>
<element name='Model' type='xsd:unsignedInt'/>
<element name='Product' type='xsd:unsignedInt'/>
<element name='BoxProduct'type='xsd:unsignedInt'/>
<element name='Httpd' type='tns:CmDevHttpd'/>
<element name='RegistrationAttempts' type='xsd:unsignedInt'/>
<element name='IsCtiControllable'  type='xsd:boolean'/>
<element name='LoginUserId' type='xsd:string'/>
<element name='Status'  type='tns:CmDevRegStat'/>
<element name='StatusReason' type='xsd:unsignedInt'/>
<element name='PerfMonObject' type='xsd:unsignedInt'/>
<element name='DChannel' type='xsd:unsignedInt'/>
<element name='Description' type='xsd:string'/>
<element name='H323Trunk' type='tns:H323Trunk'/>
<element name='TimeStamp' type='xsd:unsignedInt'/>
</sequence>
</complexType>
```

Choosing CTI Real-Time Information

Request Format

SOAP Action

This operation allows client to perform a CTI manager-related query.

The HTTP header should have following SOAP action:

```
SOAPAction: http://schemas.cisco.com/ast/soap/action/#RisPort#SelectCtiItems
```

Envelope Information

The query information should have an Envelope as follows:

```
1.  ?xml version="1.0" encoding="utf-8"?>
2.  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
3.  xmlns:tns="http://schemas.cisco.com/ast/soap/"
4.  xmlns:types="http://schemas.cisco.com/ast/soap/encodedTypes"
5.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6.  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Session ID

The following example shows a SOAP header that contains a session ID. The client sets a unique session ID.

```
7.  <soap:Header>
8.  <tns:AstHeader id="id1">
9.  <SessionId xsi:type="xsd:string">jSessionId</SessionId>
10. </tns:AstHeader>
11. </soap:Header>
12.
13. <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
14. <tns:SelectCtiItem><StateInfo xsi:type="xsd:string" /><CtiSelectionCriteria
href="#id1" /></tns:SelectCtiItem>
15. <tns:CtiSelectionCriteria id="id1" xsi:type="tns:CtiSelectionCriteria">
```

Maximum Device Information

The following example specifies the maximum number of devices that this search needs to return:

```
16. <MaxReturnedItems xsi:type="xsd:unsignedInt">10</MaxReturnedItems>
```

CTI Application/Device/Line Specification

The following example specifies on which CTI manager class Line/Device/Provider a search is provided:

```
17. <CtiMgrClass xsi:type="tns:CtiMgrClass">Line</CtiMgrClass>
```

Status of CTI Item Search

The following example specifies the Status of class on which to search:

```
18. <Status xsi:type="tns:CtiStatus">Any</Status>
```

Server Name for Search

The following example specifies the server name on which the search is performed:

```
19. <NodeName xsi:type="xsd:string" />
```

Type of Search

The following example specifies the type of selection:

```
20. <SelectAppBy xsi:type="tns:CtiSelectAppBy">AppIpAddress</SelectAppBy>
```

List of Items That Needs to be Searched

The following example specifies an array for items for which the real-time status is required:

```
21. <AppItems href="#id2" />Name /IP</tns:CtiSelectionCriteria>
22. <soapenc:Array id="id2" soapenc:arrayType="tns:SelectAppItem[2]">
23. <Item href="#id3" /><Item xsi:null="1" /></soapenc:Array>
24. <tns:SelectAppItem id="id3" xsi:type="tns:SelectAppItem">
25. <AppItem xsi:type="xsd:string"/>
26. </tns:SelectAppItem>
27. </soap:Body>
28. </soap:Envelope>
```

Response format

The Response includes a sequence of CMNodes with sequences of CTI devices and lines real-time information.

```
   <complexType name='CtiItem'>
<sequence>
<element name='AppId' type='xsd:string'/>
<element name='ProviderName' type='xsd:string'/>
<element name='UserId' type='xsd:string'/>
<element name='AppIpAddr' type='xsd:string'/>
<element name='AppStatus' type='tns:CtiStatus'/>
<element name='AppStatusReason' type='xsd:unsignedInt'/>
<element name='AppTimeStamp' type='xsd:unsignedInt'/>
<element name='CtiDevice' type='tns:CtiDevice'/>
<element name='CtiLine' type='tns:CtiLine'/>
</sequence>
    </complexType>
```

CTI Device real-time information contains the following sequence of information:

```
<complexType name='CtiDevice'>
<sequence>
<element name='AppControlsMedia'        type='xsd:boolean'/>
<element name='DeviceName'              type='xsd:string'/>
<element name='DeviceStatus'            type='tns:CtiStatus'/>
<element name='DeviceStatusReason'      type='xsd:unsignedInt'/>
<element name='DeviceTimeStamp'         type='xsd:unsignedInt'/>
</sequence>
</complexType>
```

CTI Line contains the following sequence of information:

```
<complexType name='CtiLine'>
     <sequence>
         <element name='DirNumber'        type='xsd:string'/>
         <element name='LineStatus'       type='tns:CtiStatus'/>
```

```
<element name='LineStatusReason' type='xsd:unsignedInt'/>
<element name='LineTimeStamp'    type='xsd:unsignedInt'/>
            </sequence>
       </complexType>
```

Interface to Get Server Names and Cluster Name

The interface to get cluster name `getServiceParameter`, interface to get configured servers in cluster `listProcessNodeByService`, and interface to get configured devices in cluster `listDeviceByNameAndClass` get defined as part of the AXL-DB WSDL file. Send your queries to API question mailer on these interfaces.

Authentication

The following sections describe the currently used authentication.

Basic

Basic authentication uses base64 to encode and decode the credential information. Because base64 is a two-way function, which requires no key, this protocol is a clear-text authentication that is not secure. The Basic provides an advantage because it is widely implemented by many platforms, and most HTTP client agents support it. Basic authentication can be secure if the encryption, such as SSL, gets used.

Secure

When you install/upgrade Cisco CallManager, the HTTPS self-signed certificate, httpscert.cer, automatically installs on the IIS default website that hosts the Cisco CallManager virtual directories.

Hypertext Transfer Protocol over Secure Sockets Layer (SSL), which secures communication between the browser client and the IIS server, uses a certificate and a public key to encrypt the data that is transferred over the internet. HTTPS also ensures that the user login password transports securely via the web. The following Cisco CallManager applications support HTTPS, which ensures the identity of the server: Cisco CallManager Administration, Cisco CallManager Serviceability, the Cisco IP Phone User Option Pages, the Bulk Administration Tool (BAT), TAPS, Cisco CDR Analysis and Reporting (CAR), Trace Collection Tool, and the Real Time Monitoring Tool.

For more information, refer to the Cisco CallManager Security Guide, Release 4.1.2.

Authorization

Each LDAP user gets checked against an MLA configuration for permissions. If the LDAP user in basic authentication does not have permission to "Read and Execute," the access to SOAP APIs gets denied.

Rate Control

The AXL-Serviceability Interface includes a request rate control mechanism that monitors the request rates for the complete system. If the request rate is more than supported, a "SOAP Fault" gets sent to the client and the request gets blocked. Requests rates get tracked for the system. This rate control gets enabled for "Real-time Data requests" and "Perfmon Data Requests." These rates get controlled through "Enterprise parameters" as shown in Figure 1. The system provides Help for these parameter configurations as shown in Figure 2.

Figure 1 Rate Control Enterprise Parameters

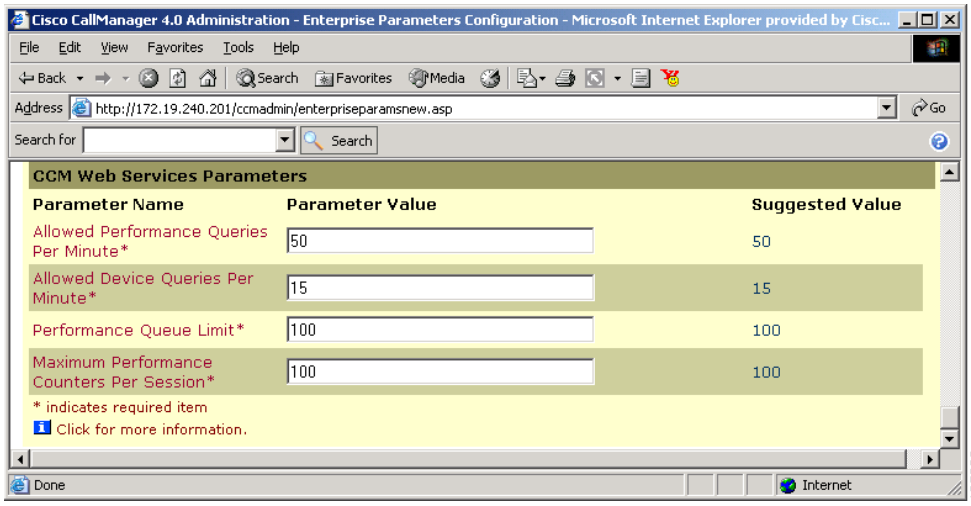
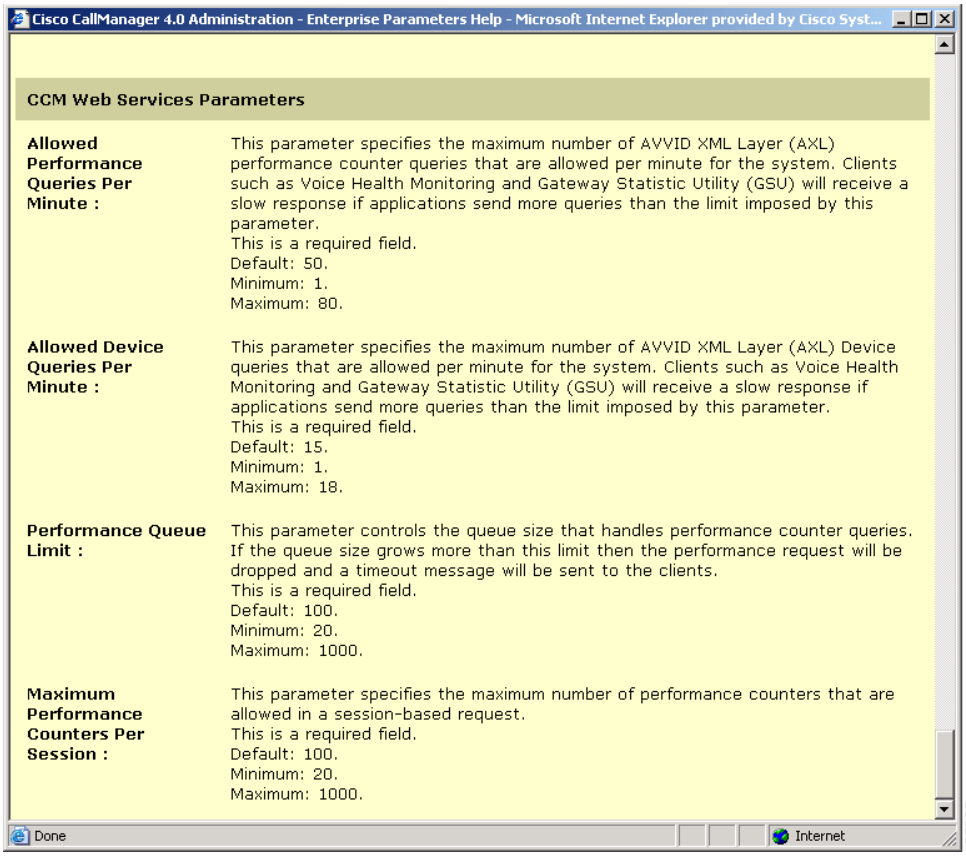


Figure 2 Help for Rate Control Parameters



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/univercd/home/home.htm

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

Ordering Documentation

You can find instructions for ordering documentation at this URL:

http://www.cisco.com/univercd/cc/td/doc/es_inpck/pdi.htm

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Ordering tool:

  http://www.cisco.com/en/US/partner/ordering/index.shtml

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco Systems Corporate Headquarters (California, USA) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

You can send comments about technical 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.

Obtaining Technical Assistance

For all customers, partners, resellers, and distributors who hold valid Cisco service contracts, Cisco Technical Support provides 24-hour-a-day, award-winning technical assistance. The Cisco Technical Support Website on Cisco.com features extensive online support resources. In addition, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not hold a valid Cisco service contract, contact your reseller.

Cisco Technical Support Website

The Cisco Technical Support 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, 365 days a year at this URL:

http://www.cisco.com/techsupport

Access to all tools on the Cisco Technical Support 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

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 automatically provides recommended solutions. If your issue is not resolved using the recommended resources, your service request will be assigned to a Cisco TAC 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 TAC 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, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

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

- The Cisco *Product Catalog* describes the networking products offered by Cisco Systems, as well as ordering and customer support services. Access the Cisco Product Catalog at this URL:

  http://cisco.com/univercd/cc/td/doc/pcat/

- *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

- *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

- 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