HP Route Analytics Management Software

Software Version: 9.20

Developer's Guide

Document Release Date: June 2012
Software Release Date: June 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Unix® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support web site at:

www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

1 Configuring and Using Queries

This chapter describes how to create queries using an Application Programming Interface (API). RAMS and RAMS Traffic queries are initiated by an Extensible Markup Language Remote Procedure Call (XML RPC).

Normally, these queries are initiated from a computer program written in a computing language such as C, Java, or Perl. This guide provides examples written in the Perl scripting language.

Initiating queries from a computer program allows you to:

- Acquire specific route analysis information.
- Integrate RAMS and RAMS Traffic products with other tools you have that support XML RPC.

To use these queries from a program, it is necessary to link in the appropriate XML RPC library or package.

XML RPC is case sensitive.

For more information, refer to **http://www.xmlrpc.com**.

Configuring RAMS and RAMS Traffic to Accept Queries

Before you can use queries, you must configure the appliance to accept queries. In a deployment with multiple Route Recorders and a centralized Modeling Engine, consider the following recommendations when you enable queries:

- For alerts and watch lists, except alerts requiring information from more than one recorder (for example, Route Change), you should enable queries on the destination Route Recorder.

- For network-wide information, enable queries on the centralized Modeling Engine.
- For information local to a recorder's area or protocol, enable queries on the Route Recorder.

To enable queries, perform the following steps:

1. From the appliance Home page, click **Administration**, and then click **Queries** on the left navigation bar.

   The Queries page opens, as shown in Figure 1.

2. The XML-RPC Query Server radio button is enabled by default so the Query Server is able to generate various reports.
3. Select **Enable Remote Access** to allow remote queries to be heard.
4. Enter a password and confirm it. The password can be from one to eight alphanumeric characters in length, is case sensitive, and must not contain nulls, blanks or underscores.
5. Click **Update**.

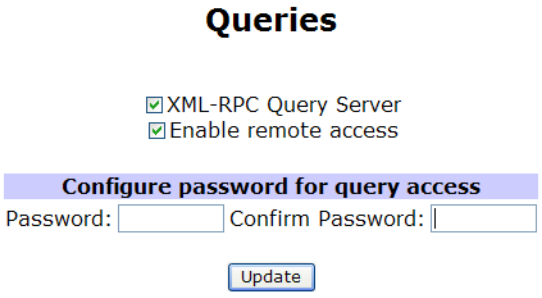

Figure 1 Queries Page

For more information about the Query Server, see Chapter 2, "High-Performance Interaction with the Query Server"

Encrypting the API Password

Encrypting the API password provides a way to preserve the integrity of the password used to make an API request. Instead of using a plain text password, the perl function creates a new encrypted password based on both the secret and the time the authentication was sent. Because the encrypted password depends on the time, it cannot be re-used.

The data is not encrypted, just the password. This allows you to only allow authorized users to query the system.

You will need to have the following Perl libraries installed before the makePassword perl code can be used:

- Digest::HMAC_SHA1
- Time::HiRes
- DateTime
- Sys::Hostname

The following is a sample implementation for the password encryption:

```
$password = makePassword('admin');
sub makePassword
   {use Digest::HMAC_SHA1 qw(hmac_sha1_hex);
    use Time::HiRes qw(gettimeofday);
    use DateTime;
    use Sys::Hostname;

    my $secret = @_[0];
    my $seconds, $microseconds, $dt, $text, $diget, $client;
    $client = hostname() . "." . $$;
    ($seconds, $microseconds) = gettimeofday;
    $dt = DateTime->from_epoch(epoch => $seconds);
    # Z is needed since we aset the time according to UTC
    $text = join(' ', "sha1", $client, $dt->iso8601 . "Z",
$microseconds);
    $digest = hmac_sha1_hex($text, $secret);
     return $text . " " . $digest;
```

Using Queries

This guide specifies the input parameters and results for the XML RPC calls listed. The *method name* for each call consists of the prefix "`RouteAnalyzer.`" plus the query name shown in the table. The queries with names beginning `api_mp_` may be used to obtain data from both IGP and BGP protocol domains. The calls with names beginning `api_vpn_` apply only to BGP/MPLS VPN protocol domains. The calls api_prefix_list_multi_origin and api_router_summarizable apply only to IGP protocol domains.

The Dumper function called by the example query programs converts XML, which uses the < and > separators and no new lines, into a more readable form. This readable form is displayed in the sample output shown throughout this *Guide*. The Dumper function is included in the standard Perl package called Data.

Table 1 XML-RPC Query Calls and Descriptions

Query	Description	Page
api_mp_close_handle	This query takes a previously generated report handle and closes the report, freeing the memory and resources it uses	4-49
api_mp_events	Lists all multi-protocol network events between two different times.	4-50
api_mp_events_handle	Lists all multi-protocol network events between two different times in chunks of a user-specified size.	4-55
api_mp_ipv6_routes	This query lists all routes including all prefix announcements from all routers announcing the prefixes, at the specified time and meeting the specified filter criteria.	4-56
api_mp_ipv6_routes_handle	This query returns a handle for all routes, including all prefix announcements from all routers announcing the prefixes at the specified time, and meeting the specified filter criteria.	4-60
api_mp_links	Lists links meeting filter criteria in a multi-protocol network.	4-61
api_mp_list_handle	Returns a user-specified number of entries starting at a user-specified point in the report	4-68
api_mp_list_paths	This query returns the total metric (if it is calculable) and the list of all paths of such cost from the source to the destination at the requested time.	4-69
api_mp_open_handle	This query lists all of the call handles that are still open, along with the types of calls they handle and their creation time.	4-86

Table 1 XML-RPC Query Calls and Descriptions (cont'd)

Query	Description	Page
api_mp_osi_routes	This query returns a handle for all routes, including all Prefix Neighbors and ES Neighbors announcements from all routers announcing the Prefix Neighbors and ES Neighbors at the specified time, and meeting the specified criteria.	4-88
api_mp_osi_routes_handle	This query returns a handle for all routes, including all Prefix Neighbors and ES Neighbors announcements from all routers announcing the Prefix Neighbors and ES Neighbors at the specified time, and meeting the specified criteria.	4-92
api_mp_prefixes_multi_origin	This query returns a list of prefixes for the specified network that are originated by more than one router (or Intermediate System in OSI terminology).	4-93
api_mp_prefixes_multi_origin_handle	This query returns a list of prefixes for the specified network that are originated by more than one router (or Intermediate System in OSI terminology).	4-99
api_mp_routers	Lists routers meeting filter criteria in a multi-protocol network.	4-120
api_mp_routers_consolidated	This query lists all the mp-level nodes present in the multi-protocol network at the specified time.	4-103

Table 1 XML-RPC Query Calls and Descriptions (cont'd)

Query	Description	Page
api_mp_routers_consolidated_handle	This query returns a handle for all the mp-level nodes present in the multi-protocol network at the specified time. For each mp-level node, the following information is provided: Sysid, Number of rt nodes, and array of rt nodes. For each rt-node, the following information is provided: router id, protocol, router type, number of interfaces for this rt node, array of interfaces ips for this router.	4-107
api_mp_routes	Lists prefix routes meeting filter criteria in a multi-protocol network.	4-108
api_mp_routes_handle	List prefix routes meeting filter criteria in a multi-protocol network in chunks of a user-specified size.	4-114
api_prefix_list_multi_orig	This query returns a list of prefixes for the specified network that are originated by more than one router.	4-120
api_resource_status	Lists the current status of the memory, disk, and swap space on the appliance.	4-124
api_router_summarizable	Lists routers advertising multiple summarizable prefixes.	4-126
api_system_health	Lists the health of all the RAMS and RAMS Traffic systems in the network, including the recording and writing status of each configured recording process and its databases.	4-129
api_unit_health	Lists the health of the specified unit, including the recording and writing status of each configured recording process and its databases, along with the locaton of the core files existing on the system.	4-133

Table 1 XML-RPC Query Calls and Descriptions (cont'd)

Query	Description	Page
api_vpn_cust_rt_list	Lists all customer names in VPN to RT (Route Target) mappings.	4-136
api_vpn_customer_pe_participation	Return statistics of participating PEs for each VPN customer.	4-139
api_vpn_customer_pe_list	Return the list of participating PEs for the specified VPN customer.	4-143
api_vpn_customer_reachability	This query returns reachability statistics for each VPN customer.	4-146
api_vpn_route_target_reachability_by_peer	Return reachability statistics at each PE for the specified VPN customer.	4-149
api_vpn_route_target_pe_participation	Return statistics of participating PEs for each route target in the specified network.	4-152
api_vpn_route_target_pe_list	This query returns the list of participating PE routers and their VPN state for the specified route target.	4-156
api_vpn_route_target_reachability	Return reachability statistics for each route target in the specified network.	4-159
api_vpn_route_target_reachability_by_peer	Return reachability statistics at each PE for the specified route target.	4-163
api_vpn_routes	Return the list of VPN routes for the specified network.	4-167
api_vpn_routes_handle	Lists all prefixes advertised in the network.	4-170

Table 2 lists the queries used for generating VPN Customer Reports. For detailed information about these queries, see Chapter 5, "VPN Customer Report Queries" in this Guide. For information about the configuration for these reports, see the "VPN Routing" chapter in *HP Route Analytics Management System User's Guide*.

Table 2 VPN Customer Reports Query Calls

Query	Description	Page
api_traffic_vpn_customer	This query returns the aggregate traffic statistics for a VPN customer.	5-14
api_traffic_vpn_customer_cos	This query returns breakdown of the aggregate traffic statistics by CoS group.	5-18
api_traffic_vpn_customer_cos_history	This query returns the history statistics (minimum, maximum, average) for the VPN customer and CoS group for a given time period.	5-12
api_traffic_vpn_customer_history	This query returns the history statistics for the VPN customer for the given time period.	5-18
api_traffic_vpn_customer_wan_connection	This query returns ingress and egress traffic statistics for traffic going to and from all the WAN connections belonging to a particular VPN customer.	5-12
api_traffic_vpn_customer_wan_connection_cos	This query returns the traffic breakdown by CoS group for each WAN connection within a given VPN customer.	5-17
api_traffic_vpn_customer_wan_connection_cos_history	This query returns the history of ingress or egress statistics for the customer, the WAN connection, and the CoS group for the given time period.	5-22
api_traffic_vpn_customer_wan_connection_history	This query retrieves the configuration of the WAN connections for a specific customer.	5-25
api_traffic_vpn_customer_wan_connection_to_wan_connection	This query returns statistics for VPN traffic between the 100 selected WAN connections.	5-29

Table 2 VPN Customer Reports Query Calls

Query	Description	Page
api_traffic_vpn_customer_wan_connection_to_wan_connection_history	This query returns the history of ingress or egress statistics for the VPN customer source and destination WAN connection provided for a given time.	5-22
api_traffic_vpn_customer_wan_connection_topn_convers	This query returns the average traffic statistics for the top 100 conversations between the source and destination addresses for a particular WAN connection.	5-26
api_traffic_vpn_customer_wan_connection_topn_dsts	This query returns the average traffic statistics for the top 100 destinations for a particular WAN connection.	5-20
api_traffic_vpn_customer_wan_connection_topn_port_protocol	This query returns traffic statistics for the top 100 protocol pairs for a particular WAN connection.	5-23
api_traffic_vpn_customer_wan_connection_topn_srcs	This query returns the traffic statistics for the top 100 sources for a particular WAN connection.	5-26
api_vpn_customer_default_reporting_wan_connections_get	This query returns a list of the WAN connections that are configured for calculating the Top N Reports, and the WAN connection to WAN connection reports.	5-20
api_vpn_customer_default_reporting_wan_connections_set	This query is used for setting a list of the WAN connections that will be used for calculating the Top N Reports and the WAN connection to WAN connection reports.	5-22
api_vpn_customer_wan_connection_get_config	This query returns the history statistics for the VPN customer for the given time period.	5-25

Table 2 VPN Customer Reports Query Calls

Query	Description	Page
api_vpn_customer_wan_connection_set_config	This query returns ingress and egress traffic statistics for traffic going to and from all the WAN connections belonging to a particular VPN customer, and for a particular period of time.	5-21
api_vpn_enabled_customer_list_get_config	This query returns statistics for VPN traffic between the 100 selected WAN connections.	5-23

Understanding Fault Codes

Table 3 lists fault codes that may be returned by XML RPC API queries. Not all values in the fault codes number space are defined or used. Fault codes in the range 1-99 correspond to standard Linux error codes; only three of these codes are used. Fault code 16 can be returned in conjunction with either of the two specified error strings.

The fault codes shown in Table 3 are used in RAMS and RAMS Traffic:

Table 3 XML RPC Fault Codes

Code	Description
16	Busy. Cannot change network topology. Please try later. Busy. Cannot change network time. Please try later.
22	Invalid argument
38	Method name <method> not recognized
200	Invalid database name.
201	Invalid topology name.
202	Invalid time.
203	Memory allocation error.
204	Incorrect password.
205	Invalid filter expression.
206	Invalid report name format.
207	Invalid report name.
208	VPN customer not a string.
209	Invalid VPN customer.
210	Invalid report handle.
211	Invalid router target format.
212	Invalid route target.

Table 3 XML RPC Fault Codes (cont'd)

Code	Description
214	Invalid IP address struct in request.
215	Invalid system ID struct in request.
216	Invalid NSAP struct in request.
217	Source router does not exist.
218	Path is of length 0.
220	Invalid management parameter.
221	Invalid trap name.
222	Unable to load routing events from database.
223	Unable to process system resource information.
224	Could not connect to one or more database(s).
225	Not permitted by license.
226	Mismatch between the supplied time range and time difference.
227	Invalid type of statistics (min/minimum, max/maximum, average/avg, or percentile) requested (case insensitive).
228	Invalid report range specified.
229	Invalid CoS.
230	Customer is not enabled for reporting.
231	No PEs found for customer.
232	Error in getting/setting default reporting sites.
233	Invalid type of data (ingress/egress) requested.
234	No VPN topologies present.
235	Invalid Address Family
236	Recording of IPv6 has been disabled

Table 3 XML RPC Fault Codes (cont'd)

Code	Description
237	Recording of OSI has been disabled
345	Edit user name could not be found.
346	Edit name could not be found
347	Failed to load edits from database.
348	Edit user name is missing.
601	IP address not found
602	Topology name not found
603	Router name not found
604	System ID not found
605	Router Group not found
606	Link Group not found
607	Path Group not found
608	IPv4 Prefix Group not found
609	IPv6 Prefix Group not found
610	Group name cannot be empty
611	Link not found
612	Not valid email id
613	Alert condition is invalid
614	Watchlist is required to configure the alert
615	Watchlist not found
616	Dispatch Specification is required to configure the alert
617	Dispatch Specification not found
618	Suppression Specification not found

Table 3 XML RPC Fault Codes (cont'd)

Code	Description
619	Alert severity is invalid
620	User is not permitted to operate on group
621	User is not permitted to operate on alerts
622	Unable to add group
623	Unable to delete group
624	Unable to find group
625	Unable to update group
626	Unable to add configuration
627	Unable to delete configuration
628	Unable to find configuration
629	Operation on tag conflicts with another operation
630	Invalid XML: tag is missing
631	Invalid network name
632	Invalid XML as per schema
633	Invalid NSAP(s)
634	Router with name, IP address and System ID not found for any protocol node
635	Delay threshold missing for path alert
636	Area name for source does not match destination
637	At least one of System ID, name, IP Address or prefix must be specified to identify a router

Table 3 XML RPC Fault Codes (cont'd)

Code	Description
638	Could not find router
639	Invalid value for field
640	Configuration schema version is incompatible with current version

Query Data Structures

There are several data structures that are used for input parameters and output results in a variety of calls. The list below distinguishes between the common structures (for example, routers, links, prefixes) that each format uses, and specifies the data types of the elements in the structures.
The list is divided into two parts: data structures for the new multi-protocol (MP) and VPN calls, and those for the older non-MP and non-VPN calls.

The order of data structure members may vary from that shown in the listings below or in the sample outputs. The program that issues the query and receives the response should put all of the data structure members into a hash table or some similar storage as they are parsed, and then reference the members from that storage.

Non-MP/VPN Data Structures

The non-MP/VPN calls use the following common structures:

- IP struct: one of the following:
  - ip4_addr: string
  - ip6_addr: string
- prefix struct:
  - masklen: int
  - ip_addr: IP struct

- router struct:
  - ip_addr: IP struct
  - maskLen: int
  - name: string
  - nodeType: string
  - nodeProto: string
  - nodeArea: string
  - nodeState: string
- interface struct:
  - source: IP struct
  - destination: IP struct
  - metric: int
  - bw: double (in Kbps)
  - delay: double (in µs)
  - state: int
- link struct:
  - source: router struct
  - destination: router struct
  - interfaces: array of interface structs

MP/VPN Data Structures

The MP/VPN calls use the following common structures:

- MP IP struct:
  - ip4_addr: string
  - ip6_addr: string (only ipv4_addr or ipv6_addr will exist at one time. Zeros are suppressed in IPv6 addresses)
- MP prefix struct:
  - masklen: int
  - ip_addr: MP IP struct
- MP state struct:
  - down: string
  - inBaseline: string (when requested)
- MP router struct:
  - mpname: string (multi-protocol router name).
  - name: string (if router name exists)
  - ipaddr: MP IP struct (if router has IP)
  - ip6addr: MP IP struct (if router has an IPv6 address)
  - type: string
  - sysid: string (if protocol is ISIS)
  - protoType: string (if protocol is ISIS)
  - overloaded: string (if protocol is ISIS)
  - model: string (if protocol is EIGRP)
  - softwareVersion: string (if protocol is EIGRP)
- MP link struct:
  - srcNode: MP router struct
  - dstNode: MP router struct
  - state: MP state struct (without baseline)
- BGP attributes struct:

- asPath: string (if attribute is present)
- origin: string (if attribute is present)
- localPref: int (if attribute is present)
- nextHop: string (if attribute is present)
- originator: string (if attribute is present)
- clusterList: string (if attribute is present)
- aggregator: (if attribute is present)
  - —ipaddr: string
  - —as: int
- atomic: bool (if attribute is present)
- extCommunities: string (if attribute is present)
- mpReachabililtyNextHop: string (if attribute is present)

- LS attributes struct:
  - metric: int
  - metricType: string
  - forwardAddr: string (if attribute is present)
- EIGRP attributes struct:
  - metricBW: int (inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)
  - metricDelay: int (in units of 10 µs * 256)
  - metricType: string
  - ifname: string (if attribute is present)
- Static attributes struct:
  - nextHops: array of
  - nextHop: string
- topology struct:
  - fullName: string
  - protocol: string

Both non-mp/vpn calls and mp/vpn calls use the following structure:

- version struct:
  - appliance_version: string
  - software_version: string

2 High-Performance Interaction with the Query Server

XML RPC is a remote procedure call mechanism which uses synchronous data transfer semantics. This means that a typical XML RPC server will convert the entire result to XML before sending the first byte of the result to the client. Similarly, a client application will typically have to wait until the XML RPC library has received the entire result before the data is delivered to the application.

Some client libraries, such as the PERL XML RPC library, impose an additional performance limitation for calls that return a large amount of data. This is due to the overly conservative buffer allocation strategy in these libraries. For example, if 4 gigabytes (GB) of data needs to be received and the default buffer size is 4 kilobytes (KB), the Perl library will allocate buffer sizes of 4, 8, 12, 16 and so on, each time copying the previous buffer on to the new buffer. This is an $O(N^2)$ computation where N is the number of bytes of data returned (in other words, very slow). A better buffer allocation strategy would be to grow the buffer size exponentially to 4, 8, 16, 32, and so forth. This is an O(N log N) computation.

The "_handle" variants of the queries improve the performance by keeping the N number small to minimize the buffer copying size.

Using a Direct TCP Connection

As an alternative to the standard XML RPC semantics, the Query Server provides a better and much faster approach. The Query Server does not wait to form the entire result in memory before starting to send the result to the user. Instead, the data is streamed packet-by-packet as the data is generated to fill each packet. In the client, the alternative method is to skip using an XML RPC

library and instead treat the data exchange as XML-over-HTTP-over-TCP. This allows the application to receive the data as each packet arrives rather than having the library accumulate the full result into a buffer first.

This method eliminates the need to use the "_handle" variant of the queries to keep the buffer small. In fact, with this method one large query is faster than breaking the request into a sequence of small queries using the "_handle" variant.

The following is an example of a direct TCP connection using a sample perl program:

```
#!/usr/bin/perl
use IO:Socket;
$remote = IO::Socket::INET->new(Proto => "tcp",
                  PeerAddr => "localhost",
                  PeerPort => "2000");

$remote->autoflush(1);
$EOL = "\015\012";
print $remote "<methodCall>
 <methodName>RouteAnalyzer.api_mp_routes</methodName>
 <params>
  <param>
   <value><string>admin</string></value>
  </param>
 <param>
  <value><string>UCBJul03a</string></value>
  </param>
 <param>
   <value><dateTime.iso8601>20030723T18:12:09Z</
dateTime.iso8601></value>
  </param>
  <param>
    <value><string>any</string></value>
  </param>
 </params>
</methodCall>" . $EOL;
while (<$remote>) { print; }
close $remote;
```

The above example skips sending the HTTP POST header before the XML RPC function call. The HTTP header is optional, and is ignored by the Query Server. The output from the server will contain a simple HTTP header for compatibility with XML RPC syntax, but this header may be ignored by the client.

Keeping the Connection Open

XML RPC uses a new TCP (HTTP) connection for each request. If you need to make many requests back-to-back, you can avoid the overhead of repeated connection establishment by keeping the connection open. However, you need to declare this to the server by issuing the following function call as your first request:

```
api_conn_keep_open("password")
```

You have one hour to issue your next request. If you keep the connection open without issuing any requests for ten minutes, the server will close the connection. To close the connection explicitly by the server, use the following request:

```
api_conn_close("password")
```

Alternatively, you can close the connection at the client end.

Once api_conn_keep_open("password") is authenticated, the subsequent queries on this connection are not subject to password checking; however, at least an empty password must be passed.

The following example illustrates keeping the connection open for multiple queries:

```
#!/usr/bin/perl
use IO::Socket;
$remote = IO::Socket::INET->new(Proto => "tcp",
PeerAddr => "localhost",
PeerPort => "2000");
$remote->autoflush(1);
```

```
print $remote"<methodCall>
  <methodName>RouteAnalyzer.api_conn_keep_open</methodName>
  <params>
   <param>
    <value><string>admin</string></value>
    </param>
   </params>
 </methodCall>";
while (<$remote>) { print; last if (/\/methodResponse/); }
print $remote "<methodCall>
 <methodName>RouteAnalyzer.api_mp_routes</methodName>
 <params>
  <param>
   <value><string></string></value>
 </param>
 <param>
  <value><string>UCBJul03a</string></value>
</param>
<param>
 <value><dateTime.iso8601>20030723T18:12:09Z</
dateTime.iso8601></value>
 </param>
 <param>
  <value><string>any</string></value>
</param>
<param>
 <value><int>1</int></value>
  </param>
 </params>
</methodCall>";
while (<$remote>) { print; last if (/\/methodResponse/); }
print $remote "<methodCall>
 <methodName>RouteAnalyzer.api_mp_routes</methodName>
  <params>
   <param>
    <value><string></string></value>
   </param>
  <param>
   <value><string>UCBJul03a</string></value>
 </param>
 <param>
  <value><dateTime.iso8601>20030723T18:12:09Z</
```

```
dateTime.iso8601></value>
   </param>
   <param>
         <value><string>any</string></value>
   </param>
   <param>
    <value><int>1</int></value>
   </param>
  </params>
</methodCall>";
while (<$remote>) { print; last if (/\/methodResponse/); }
print $remote "<methodCall>
 <methodName>RouteAnalyzer.api_conn_close</methodName>
 <params>
   <param>
    <value><string></string></value>
   </param>
 </params>
</methodCall>";
while (<$remote>) { print; last if (/\/methodResponse/); }
close $remote;
```

Cutting Down XML RPC Overhead

XML RPC transported data is verbose due to spelling out of the type for each piece of information. The Query Server can reduce this overhead by approximately 75% by delivering the data in an alternative XML format that is more concise but not compatible with the XML RPC specification. This speeds up total transfer time significantly for slow network connections.

To select the alternative format, you must first issue the `api_conn_keep_open` command, then issue the following command:

```
api_conn_brief_xml("password")
```

This command takes effect only for the current connection. If you close the connection and open another one, you must repeat these commands.

When this option is used, the structure members are encoded in a compressed XML format. Before this transformation, a structure member foo whose value is bar, is encoded as the following :

```
<member>
 <name>foo</name>
 <value><type of bar>bar</type of bar></value>
</member>
```

With this option, it is encoded as:

```
<foo>bar</foo>
```

Note that the type of bar is also omitted. For example, the following tags:

```
<string>
<i4>
<double>
<boolean>
<dateTime.iso8601>
```

are omitted if bar is a string, integer, double, boolean, or time, respectively. If bar is an array, both <array> and <data> tags are omitted. If bar is a structure, then the <struct> tag is omitted, and the members of the structure is transformed recursively using this procedure.

Note that this definition is recursive. A structure containing an array of structures will have all of its tags removed. The typical output from the appliance is like this, and all of the extraneous tags will be removed. This simple transformation reduces the size of the output significantly. However, the client is now expected to know the type of each of the members a priori.

Concurrency

The Query Server will allow up to 16 concurrent connections, provided that they all contain the same values for the input parameters {database name} and {time}. All other requests will return the Server Busy error (C error code EBUSY) until the previous requests are completed.

This limited concurrency will be most useful in scenarios where the multiple requests can be coordinated, such as in a sophisticated client program that can benefit from opening multiple connections asynchronously. Below are tips to enable concurrent queries:

- Some client implementations will request the most recent data by specifying a {time} parameter slightly in the future so that the server will reduce this to the current time. Since the queries are issued at slightly different times, the current time value may have been updated from one query to the next, causing the {time} parameter to no longer match.

Instead, all of the concurrent queries should specify the same {time} parameter just slightly in the past (relative to the server's current time), to the closest 5 minute boundary.

- If you need to make multiple queries that need different databases within the complete network topology, you can still issue the queries concurrently if they all specify the complete topology for the {database} parameter, and then use the {filter} parameter to restrict the results to just the database(s) desired.

To make such programming easier, the following command can be used to learn what {database name} and {time} parameters were specified for the topology that the Query Server currently has loaded:

```
api_get_topology("password")
```

A sample output of this command is as follows (shown in the more concise format selected by the api_conn_brief_xml command):

```
<topology>
    <network_name>PDLabDualAS</network_name>
    <time>2003-06-26T01:10:09</time>
    <busy>0</busy>
     </topology>
```

This example demonstrates that the server has PDLabDualAS topology loaded at Jun 26 2003 01:10:09. It also indicates that the Query Server is not busy, so the next query (either on a connection that is still open or on a new connection) can change the {database name} and/or {time} parameters without getting a "server busy" error.

Blocking Queries

Even though the Query Server allows up to 16 concurrent queries, a query may block for a short time before returning any output (including just an error such as EBUSY). This is because database operations inside the Query Server are synchronous. If one query requires loading lots of data from the database, any concurrent queries meanwhile would block. Once the database operation finishes, the other queries would proceed without waiting for the transmission of the result to the first querier. Often the transmission time is much larger than the database loading time. The maximum expected blocking time would be tens of seconds.

Blocked queries are held in the operating system's socket buffer queue. Once the blocked query executes, it may succeed or return an error.

The Query Server returns EBUSY when the topology cannot be changed, instead of further blocking the query, so the client has control over how to proceed. The user may have a pool of Query Servers and may prefer to issue the query to a different server, or simply wait for a few seconds, and then re-issue the query to the same server.

Handling Topology and Time Changes

Changing the topology and time at the Query Server takes very little time. The performance of the server degrades if each query requests a different model. For example, using a scenario where two users are making continuous queries that require different topology parameters, they can improve performance by issuing multiple queries from one user, and then switching to the other user. If the difference is only in selecting the portion of the complete topology is relevant (OSPF or BGP), the two users will get the best performance if they both request to load both the OSPF and BGP databases and then use filters to restrict the result to just the OSPF or BGP portion, respectively.

Another performance suggestion is that time movement is often faster moving forward than by moving backward. If you need to make a series of queries but at different topology times, it is best to do them in increasing order of time.

3 Using Re-Entrant Queries

The following queries (api_mp_events, api_mp_routes, and api_vpn_routes) have the following re-entrant versions:

- api_mp_events_handle
- api_mp_osi_routes_handle
- api_mp_prefixes_multi_origin_handle
- api_mp_routers_consolidated_handle
- api_mp_routes_handle
- api_vpn_routes_handle

These versions allow you to create a large report, retrieve it in pieces, and then close it at a later time. This is contrary to the standard way of retrieving a report, which creates the report, retrieves all entries, and then closes the report all in one call. For reports like these that may take longer and consume a large amount of resources, splitting the report into a series of smaller calls avoids having one call monopolize the appliance for an extended period of time.

To use re-entrant queries, perform the following steps:

1. Use the re-entrant version of the report to create the report and return its handle.

   The handle will be an integer used to identify the report in later calls.

   The parameters for the re-entrant versions of these calls are the same as the standard versions, with the exception of {max entries}. Another difference is the handle is returned instead of the entire report.

2. Use the api_mp_list_handle call to return a user-specified number of entries starting at a user-specified entry in the report.

   Typically, you can start at the beginning and return equal-sized chunks of the report until the entire report has been retrieved, but there is no restriction to retrieving any sequential chunk of the report that the you desire.

3 Use api_mp_close_handle when you are done to close the report and to release any resources it may be using.

Once this is done, you can no longer use api_mp_list_handle to retrieve pieces of the report.

In the following example, all three calls are combined so that the user can retrieve the results of the api_mp_events call in chunks of 500 entries at a time:

Example

```
#!/usr/bin/perl

if (!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_events_handle ip database [filter] \n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("4 Jul 2000 00:55:17 PST");
my $t2 = str2time("4 Jul 2005 11:55:18 PST");

# 10K entries default; if -1 entered, RPC implementation will return all
my $num = 10000;
$num = $ARGV[3] if ($#ARGV >= 3);

my $openreq = RPC::XML::request->new(
'RouteAnalyzer.api_mp_events_handle',
RPC::XML::RPC_STRING($password),
RPC::XML::RPC_STRING($database),
RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
RPC::XML::datetime_iso8601->new(time2iso8601($t2)),
RPC::XML::RPC_STRING($filter),
RPC::XML::RPC_INT($num)
);
```

```
my $openres = $client->send_request($openreq);
if ($openres->is_fault) {print("---XMLRPC FAULT ---"); }
my $overall = $openres->value;

my $handle = int($overall->{result});
my $total = int($overall->{numRequestedEntries});

# chunk size is set to 5000
my $step = 5000;

my $index;
my $num;
my $result;
for ($index = 0; $index < $total; $index += $step) {
my $delta = $total - $index;
if ($delta > $step) { $delta = $step; }

my $listreq = RPC::XML::request->new(
                    'RouteAnalyzer.api_mp_list_handle',
                     RPC::XML::RPC_INT($handle),
                    RPC::XML::RPC_STRING($database),
                     RPC::XML::RPC_INT($index),
                     RPC::XML::RPC_INT($delta),
                  );

my $listresp = $client->send_request($listreq);
        for (@{$listresp->value->{result}}) {
push @{$result}, $_;
}
}

my $closereq = RPC::XML::request->new(
'RouteAnalyzer.api_mp_close_handle',
RPC::XML::RPC_INT($handle),
RPC::XML::RPC_STRING($database),
);

my $closeres = $client->send_request($closereq);


$overall->{result} = $result;
my $p = Dumper($overall);
print $p;
```

4 XML RPC Queries

This chapter describes the calls, input parameters and results for XML RPC queries:

- api_load_topology
- api_mp_close_handle
- api_mp_events
- api_mp_events_handle
- api_mp_ipv6_routes
- api_mp_ipv6_routes_handle
- api_mp_links
- api_mp_list_all_paths
- api_mp_list_handle
- api_mp_list_paths
- api_mp_open_handle
- api_mp_osi_routes
- api_mp_osi_routes_handle
- api_mp_osi_routes_handle
- api_mp_prefixes_multi_origin
- api_mp_prefixes_multi_origin_handle
- api_mp_routers
- api_mp_routers_consolidated
- api_mp_routers_consolidated_handle
- api_mp_routes
- api_mp_routes_handle

- api_prefix_list_multi_orig
- api_resource_status
- api_router_summarizable
- api_system_health
- api_unit_health
- api_unload_topology
- api_vpn_cust_rt_list
- api_vpn_customer_pe_participation
- api_vpn_customer_pe_list
- api_vpn_customer_reachability
- api_vpn_route_target_reachability_by_peer
- api_vpn_route_target_pe_participation
- api_vpn_route_target_pe_list
- api_vpn_route_target_reachability
- api_vpn_route_target_reachability_by_peer

api_load_topology

RPC Call: RouteAnalyzer.api_load_topology {password} {database name} {time} {name of set of edits} {edit user} {continueOnError}

This query loads the specified topology and processes the routing events up to the specified time. If a set of planning edits is specified, the query then switches to Planning mode and applies the edits.

This API can be used only in persistent connection mode, in which the first call is api_conn_keep_open. After the api_load_topology call is made, the topology enters a locked state in which subsequent API calls to change the loaded topology are disallowed. The topology remains locked until one of the following conditions is satisfied:

- The persistent connection is closed. In this case all applied edits are removed. The topology is unlocked and becomes able to accept other API calls.

- Another request to api_load_topology is issued in the same persistent connection. Edits that were applied in the previous api_load_topology are removed, and the topology is updated with the edits from the new api_load_topology call.
- api_unload_topology is issued in the same persistent connection. This removes all applied edits. The topology is unlocked and becomes able to accept other API calls.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as autonet, which includes the subtree below it, or a complete database name, such as autonet.pdinet.ISIS.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. The query results will be calculated based on the network state at the specified time.
- **name of a set of edits** – An optional parameter identifying a set of edits that were saved in the GUI with the specified name.
- **edit user** – The account name of the user who saved the set of edits. This parameter must be included if a set of edits is specified.
- **continueOnError** – An optional boolean flag to control what action to take if an error occurs in applying one of the edits. If true, application of the subsequent edits will continue. If false (the default), application of the edits will stop.

Structure of Output

This query returns a fault code for hard errors such as when the name of the set of edits is unknown, or if this call is issued on a connection that is not persistent. When the input parameters are all valid so the topology is loaded and the edits are applied, either all successfully or with some edit errors, then a message is returned to indicate whether any edit errors occurred. In that case, the structure of the output is as follows:

- vinfo: version struct
- result: string containing the message for success or number of errors and error message

Sample Output

The following example shows the result when all edits are applied successfully:

```
{
  'vinfo' => {
```

```
      'software_version' => '9.0.30-R Traffic Explorer',
      'appliance_version' => '9.0.30-R'
   },
   'result' => 'Successfully applied edits to topology.',
}
```

The following example shows the result when some errors are encountered while applying edits (where continueOnError is true so the count of errors can be more than one):

```
{
   'vinfo' => {
      'software_version' => '9.0.30-R Traffic Explorer',
      'appliance_version' => '9.0.30-R'
   },
   'result' => '6 error(s) encountered while applying edits. Error 1:
10.120.1.13 is not
   a valid router name.',
}
```

api_load_topology_edits

RPC Call: RouteAnalyzer.api_load_topology_edits {password} {editString} {continueOnError}

This query switches to Planning mode and loads the edit or set of edits that has been supplied in the edit language. You must first call api_load_topology and load the database on which edits need to be applied. Calling the api_load_topology_edits will not unload the edits that were loaded in previous call of api_load_topology_edits (if any). Calling api_unload_topology can unload the edits.

Input Parameters

- **password** – The password configured for queries.
- **editString** – Edit or a set of edits in the edit language.
  Example: add router - area PDI -proto bgp -rtrID 11.11.11.1 -x -1306.03 -y 1393.63
- **continueOnError** – An optional boolean flag to control what action to take if an error occurs in applying one of the edits. If true, application of the subsequent edits will continue. If false (the default), application of the edits will stop.

Structure of Output

This query either returns an error code indicating what the error is or a message that edits have successfully applied:

- vinfo: version struct
- result: string containing the message for success or number of errors and error message

Example

```
#!/usr/bin/perl
use IO::Socket;
use RPC::XML 'time2iso8601';
use Date::Parse;

$remote = IO::Socket::INET->new(Proto => "tcp",
PeerAddr => "localhost",
PeerPort => "2000");
$remote->autoflush(1);
my $t1 = str2time("01 Aug 2011 14:06:28");
my $t11 = time2iso8601($t1);

print $remote "<methodCall>
  <methodName>RouteAnalyzer.api_conn_keep_open</methodName>
  <params>
   <param>
    <value><string>admin</string></value>
    </param>
   </params>
 </methodCall>";

while (<$remote>) { print; last if (/\/methodResponse/); }

print $remote "<methodCall>
  <methodName>RouteAnalyzer.api_conn_brief_xml</methodName>
  <params>
   <param>
    <value><string>admin</string></value>
    </param>
   </params>
 </methodCall>";
while (<$remote>) { print; last if (/\/methodResponse/); }
```

```
print $remote "<methodCall>
<methodName>RouteAnalyzer.api_load_topology</methodName>
 <params>
  <param>
   <value><string></string></value>
 </param>
 <param>
  <value><string>PDI</string></value>
</param>
<param>
 <value><dateTime.iso8601>$t11</dateTime.iso8601></value>
 </param>
 </params>
</methodCall>";

while (<$remote>) { print; last if (/\/methodResponse/); }

print $remote "<methodCall>
 <methodName>RouteAnalyzer.api_load_topology_edits</methodName>
 <params>
  <param>
   <value><string></string></value>
 </param>
 <param>
  <value><string>add router -area PDI.Dynalab.BGP/AS65535 -proto bgp -rtrID
11.11.11.1 -x -1306.03 -y 1393.63</string></value>
</param>
 <param>
 <value><boolean>false</boolean></value>
</param>
</params>
</methodCall>";

while (<$remote>) { print; last if (/\/methodResponse/); }
```

Sample Output

The following example shows the result when all edits are applied successfully:

```
{
}
```

```
'vinfo' => { 'software_version' => '9.0.30-R Traffic Explorer',
'appliance_version' => '9.0.30-R'
},
'result' => 'Successfully applied edits to topology.',
```

The following example shows the result when some errors are encountered while applying edits (where continueOnError is true so the count of errors can be more than one):

```
{
}
'vinfo' => { 'software_version' => '9.0.30-R Traffic Explorer',
'appliance_version' => '9.0.30-R'
}, 'result' => '6 error(s) encountered while applying edits. Error 1:
10.120.1.13 is not a valid router name.',
```

api_mp_close_handle

RPC Call: RouteAnalyzer.api_mp_close_handle {handle} {database}

This query takes a previously generated report handle and closes the report, freeing the memory and resources it uses. After this occurs, the report handle can no longer be used by RouteAnalyzer.api_mp_list_handle.

Input Parameters

- **handle** – An integer previously generated by an RPC call ending in _handle.
- **database** – One or more space-separated names in the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

Structure of Output

- vinfo: version struct
- numReturned Entries: int
- network_name: string
- report_time: ISO 8601 UTC time

- totalEntries: int
- result: blank string

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample output details.

api_mp_events

RPC Call: RouteAnalyzer.api_mp_events {password} {database name} {time t1} {time t2} {filter} {max entries} {show protocol packets}

This query lists all multi-protocol network events between times t1 and t2 that meet the specified filter criteria. Examples of events include BGP prefixes announced or withdrawn and IGP adjacencies added or dropped.

The query can return a large number of BGP events in a small amount of time. You can keep the number of events manageable by refining your filter and shortening the time period. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all BGP events within times t1 and t2. Alternatively, you can supply the optional {max entries} parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time t1, t2** – Two times specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will include events that occurred between the two specified times.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

- **max entries** – An optional 32-bit integer parameter specifying the maximum number of entries to return in the query. The default value is -1, meaning no limit.
- **show protocol packets** – An optional boolean parameter to control whether protocol packet events are included in the returned list. The default value is false.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following structures:
  - target: string
  - attributesText: string (if not BGP)

— time :

- seconds: int
- useconds: int

— topology: topology struct

— operation: string

— router: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_events ip database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("05 Nov 2004 11:09:04 PST");
my $t2 = str2time("05 Nov 2004 11:53:52 PST");

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_events',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::datetime_iso8601->new(time2iso8601($t2)),
```

```
    RPC::XML::RPC_STRING($filter), 150 ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);

}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
},
  'numReturnedEntries' => '150',
  'network_name' => 'baklab701',
  'report_time' => '20051107T23:13:37',
  'totalEntries' => '93971',
  'result' => [
    {
      'target' => '',
      'attributesText' => 'Type: Internal Router',
      'time' => {
        'seconds' => '1130545402',
        'useconds' => '966898'
      },
      'topology' => {
        'fullName' => 'baklab701.OSPF/0.0.0.1',
        'protocol' => 'OSPF'
      },
      'operation' => 'Drop Router',
      'router' => '192.168.0.87'
    },
    {
      'target' => '192.168.0.87',
      'attributesText' => 'Metric: Down',
      'time' => {
        'seconds' => '1130545402',
        'useconds' => '966898'
      },
      'topology' => {
        'fullName' => 'baklab701.OSPF/0.0.0.1',
        'protocol' => 'OSPF'
      },
      'operation' => 'Drop Neighbor',
      'router' => '192.168.0.2 DR'
    },
    ....
  ]
```

```
}
```

api_mp_events_handle

RPC Call: RouteAnalyzer.api_mp_events {password} {database name} {time t1} {time t2} {filter} {show protocol packets}

This query returns a handle for all multi-protocol network events between times t1 and t2 that meet the specified filter criteria. Examples of events include BGP prefixes announced or withdrawn and IGP adjacencies added or dropped.

The query can return a large number of BGP events in a small amount of time. You can keep the number of events manageable by refining your filter and shortening the time period. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all BGP events within times t1 and t2.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time t1, t2** – Two times specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will include events that occurred between the two specified times.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **show protocol packets** – An optional boolean parameter to control whether protocol packet events are included in the returned list. The default value is false.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: int

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample details.

api_mp_ipv6_routes

RPC Call: RouteAnalyzer.api_mp_ipv6_routes {password} {database name} {time} {filter} {max entries}

This query lists all routes including all prefix announcements from all routers announcing the prefixes, at the specified time and meeting the specified filter criteria.

The query can return a large number of BGP routes in a small amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional {max entries} parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.AS64600.BGP/AS64600/IPv6.

- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **max entries** – An optional 32-bit integer parameter specifying the maximum number of entries to return in the query.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - topology: topology struct
  - attributes: LS attribute struct (if it is LS)
  - attributes: EIGRP attribute struct (if it is EIGRP)
  - attributes: string (if other IGP)
  - attributes: Static attribute struct (if it is Static)
  - attributes: BGP: attribute struct (if it is BGP)
  - attributes: string (if other)
  - prefix6: string
  - router: MP router struct
  - state: MP state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
```

```
    printf "usage: RouteAnalyzer.api_mp_routes ip database filter\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = time;

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_routes',
             RPC::XML::RPC_STRING($password),
             RPC::XML::RPC_STRING($database),
             RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
             RPC::XML::RPC_STRING($filter), 150));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => 246,
  'network_name' => 'PacketDesignIPv6.AS64600',
  'network_time' => '20090130T14:15:39',
  'report_time' => '20090130T14:15:39',
  'totalEntries' => '246',
  'result' => [
    {
      'prefix6' => '2008:bbbb:12::/126',
      'topology' => {
        'fullName' => 'PacketDesignIPv6.AS64600.bgp.ip6.BGP/AS64600/IPv6',
        'protocol' => 'BGP'
      },
      'attributes' => {
        'mpReachabilityNextHop' => '::ffff:172.16.1.1',
        'origin' => 'INCOMPLETE',
        'localPref' => '100',
        'asPath' => '',
        'med' => '0'
      },
      'router' => {
        'name' => 'R1',
        'type' => 'IBGP Peer',
        'ipaddr' => '172.16.1.1'
      },
      'state' => {
        'inBaseline' => 'true',
        'down' => 'false'
      }
    },
    {
      'prefix6' => '2008:bbbb:12::/126',
      'topology' => {
        'fullName' => 'PacketDesignIPv6.AS64600.ISIS/Level2',
        'protocol' => 'ISIS'
      },
```

```
        'attributes' => {
          'metric' => '10',
          'metricType' => 'Internal'
        },
        'router' => {
          'overloaded' => 'false',
          'sysid' => '49.0100.1760.1600.1001.00',
          'name' => 'R1',
          'type' => 'L1L2 Router',
          'protoType' => 'IPv4 + IPv6',
          'ipaddr' => '172.16.1.1',
          'ip6addr' => '2008:bb01::1'
        },
        'state' => {
          'inBaseline' => 'false',
          'down' => 'false'
        }
      },
....

  ]
}
```

api_mp_ipv6_routes_handle

RPC Call: RouteAnalyzer.api_mp_routes {password} {database name} {time} {filter}

This query returns a handle for all routes, including all prefix announcements from all routers announcing the prefixes at the specified time, and meeting the specified filter criteria.

The query can return a large number of BGP routes in a small amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional {max entries} parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.AS64600.BGP/AS64600/IPv6.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: int

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample output details.

api_mp_links

RPC Call: RouteAnalyzer.api_mp_links {password} {database name} {time} {filter}

This query lists all network links present in the multi-protocol network at the specified time. The results may be filtered to select only the links connected to a single router, for example. The output consists of information about the source node, the destination node, and the link between them.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following structures:
  - sif: string (if IGP or static)
  - dif: string (if IGP or static)
  - metric: int (if IGP)
  - bw: double (in Kbps, if EIGRP or static)
  - delay: double (in µs, if EIGRP or static)
  - configured_bw: double (if configured)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_links ip database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("05 Nov 2004 02:11:27 PST");

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_links',
             RPC::XML::RPC_STRING($password),
             RPC::XML::RPC_STRING($database),
             RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
             RPC::XML::RPC_STRING($filter)));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '150',
  'network_name' => 'LabRight',
  'report_time' => '20050725T23:40:42',
  'totalEntries' => '150',
  'result' => [
    {
      'link' => {
        'srcNode' => {
          'type' => 'Traffic Explorer',
          'ipaddr' => '192.168.122.90'
        },
        'dstNode' => {
          'type' => 'IBGP Peer',
          'ipaddr' => '192.168.100.100'
        },
        'state' => {
          'down' => 'false'
        }
      },
      'topology' => {
        'fullName' => 'LabRight.ConfedsTest.ConfedTestTop.BGP/AS65510',
```

```
        'protocol' => 'BGP'
      }
    },
   {
      'link' => {
        'srcNode' => {'type' => ..., 'ipaddr' => ...},
        'dstNode' => {'type' => ..., 'ipaddr' => ...},
        'state' => {'down' => ...}, //end of link
      'dif' => ...,
      'sif' => ...,
      'topology' => {'fullName' => ..., 'protocol' => ...}
    }, //end of topology
    'metric' => ...
    'configured_bw' => ...
    }
  ]
}
```

api_mp_list_all_paths

RPC Call: RouteAnalyzer.api_mp_list_all_paths {password} {database name} {time} [filter]

This query lists paths for all source/destination pairs of all the routers that are available in network topology. It is an extension to the api_mp_list_paths query that iterates over the N*N combinations of all router pairs in the network as the source and destination of the path.

See also api_mp_list_paths on page 69.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **filter** – A optional filter expression that limits the results to the subset that matches the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

For this query, the most helpful filters are those on the path source and destination, and those can be constructed through the use of filter expressions. For example, an expression of type "((pathSource rtr1) or (pathSource rtr2) or (pathSource rtr3)) and (pathDestination rtr4) or (pathDestination rtr5) or (pathDestination rtr6))" can be used to limit the query to a specific subset of source and destination routers. You can combine filters using the rules that are described in the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide*.

Structure of Output

- vinfo: version struct
- numReturnEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following structures:
  - path_src: router struct
  - path_dst: prefix IP struct
  - path_cost: int
  - paths: array of the following (this will list all ECMP paths between the given source and destination):

    —path: string

    —cost: int

    —num_hops: int

    —path_hops: array of the following:

    —hops_src: router struct

    —hop_dst: router struct

—area/AS: string (if it is applicable)

—interfaces: array of the following:

—sif: either MP IP struct or string “Unnumbered” for IPv4 IGP, or int interface ID for OSPFv3, or string “Not applicable” for OSI

—dif: either MP IP struct or string “Unnumbered” for IPv4 IGP, or int interface ID for OSPFv3, or string “Not applicable” for OSI

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP; in units of 10 µs * 256)

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP; in units of 10 µs * 256)

—metric: int (if IGP)

—protocol: string

—prefix: prefix struct

—lookups: array of the following

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP)

—metric: int (if IGP)

—protocol: string

—prefix: prefix struct

—tunnel_hops: array of the following

—hop_src: router struct

—hop_dst: router struct

—sif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP (applicable for non-FRR hop)

—dif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP (applicable for non-FRR hop)

—frr_hops: array of the following

—hop_src: router struct

—hop_dst: router struct

—sif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP

—dif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP

Example and Sample Output

Refer to the sample output for api_mp_list_paths on page 69 to see the output format.

api_mp_list_handle

RPC Call: RouteAnalyzer.api_mp_list_handle {handle} {database} {index} {delta}

This query takes a previously generated report handle and returns a user-specified number of entries starting at a user-specified point in the report.

Input Parameters

- **handle** – An integer previously generated by an RPC call ending in "_handle."
- **database** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **index** – The entry in the report to start returning information.
- **delta** – The number of entries to return information for.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int

- result: depends on the report being called.

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample output details.

api_mp_list_paths

RPC Call: RouteAnalyzer.api_mp_list_paths {password} {database name} {source address} {dest prefix} {time}

This query returns the total metric (if it is calculable) and the list of all paths of such cost from the source to the destination at the requested time. Each path contains information on that path and a description of each hop in the path.

See also api_mp_list_all_paths on page 65.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **source address** – An XML struct that contains the router ID or a router interface address as an IPv4 address and a mask length of 32. The mask address is included for backward compatibility, but this query ignores it. For the OSI network, the source address is an XML struct that contains the system ID of the intermediate system. For IPv6, an XML struct that contains the IPv6 address of router and mask length of 128. For the OSPFv3 network, the source address must still be an IPv4 address, usually the router ID.
- **dest prefix** – An XML struct that contains any destination prefix consisting of an IPv4 address, such as `192.168.123.125`, and a mask length, such as 27. For the OSI network, the dest prefix is the NSAP. For IPv6, an XML struct that contains any destination prefix consisting of an IPv6 address, such as `2001:a221::21`, and a mask length, such as 64.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

Structure of Output

- vinfo: version struct
- numReturnEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following structures:
  - path_src: router struct
  - path_dst: prefix IP struct
  - path_cost: int
  - paths: array of the following (this will list all ECMP paths between the given source and destination):

    —path: string

    —cost: int

    —num_hops: int

    —path_hops: array of the following (pseudonodes are skipped):

    —hops_src: router struct

    —hop_dst: router struct

—area/AS: string (if it is applicable)

—interfaces: array of the following:

—sif: either MP IP struct or string “Unnumbered” for IPv4 IGP, or int interface ID for OSPFv3, or string “Not applicable” for OSI

—dif: either MP IP struct or string “Unnumbered” for IPv4 IGP, or int interface ID for OSPFv3, or string “Not applicable” for OSI

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP; in units of 10 µs * 256)

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP; in units of 10 µs * 256)

—metric: int (if IGP)

—protocol: string

—prefix: prefix struct

—lookups: array of the following

—bw: int (if EIGRP; inverse of bandwidth, in bps, scaled by $2.56 * 10^{12}$)

—delay: int (if EIGRP)

—metric: int (if IGP)

—protocol: string

—prefix: prefix struct

—tunnel_hops: array of the following

—hop_src: router struct

—hop_dst: router struct

—sif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP (applicable for non-FRR hop)

—dif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP (applicable for non-FRR hop)

—frr_hops: array of the following

—hop_src: router struct

—hop_dst: router struct

—sif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP

—dif: either MP IP prefix struct, or string "Unnumbered" for IPv4 IGP

Example

```
#!/usr/bin/perl
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
my $t1 = str2time("2 Feb 2009 17:18:21 PST");
push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_list_paths',
                RPC::XML::RPC_STRING($password),
                RPC::XML::RPC_STRING($database),
                ##### source #####
                new RPC::XML::struct(ip_addr =>
                        new RPC::XML::struct(ip6_addr => "2001:a331::31"),
                        masklen => 128),
                ##### destination #####
                new RPC::XML::struct(ip_addr => new
RPC::XML::struct(ip6_addr => "2002:cccc::"),
                        masklen => 64),
                RPC::XML::datetime_iso8601->new(time2iso8601($t1)))
                );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

```
}
```

Sample Output (for IPv4)

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => 'unknown appliance version'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'lab',
  'network_time' => '20080902T01:18:21',
  'report_time' => '20090428T04:47:10',
  'totalEntries' => '1',
  'result' => [
    {
      'path_dst' => {
        'masklen' => '32',
        'ip_addr' => {
          'ip4_addr' => '57.57.57.57'
        }
      },
      'path_cost' => '-1',
      'path_src' => {
        'type' => 'AS Border Router',
        'ipaddr' => {
          'ip4_addr' => '24.0.0.12'
        }
      },
      'paths' => [
        {
          'cost' => '-1',
          'path_hops' => [
            {
              'hop_src' => {
                'type' => 'AS Border Router',
                'ipaddr' => {
                  'ip4_addr' => '24.0.0.12'
                }
              },
              'protocol' => 'BGP',
```

```
  'hop_dst' => {
    'type' => 'LAN Pseudo-Node',
    'ipaddr' => {
      'ip4_addr' => '192.168.101.2'
    },
    'prefix' => {
      'masklen' => '24',
      'ip_addr' => {
        'ip4_addr' => '192.168.101.0'
      }
    }
  },
  'area/AS' => 'lab.BGP/AS65522',
  'lookups' => [
    {
      'protocol' => 'OSPF',
      'metric' => '2',
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '192.168.127.0'
        }
      }
    }
  ],
  'prefix' => {
    'masklen' => '24',
    'ip_addr' => {
      'ip4_addr' => '57.57.57.0'
    }
  }
},
{
  'hop_src' => {
    'type' => 'LAN Pseudo-Node',
    'ipaddr' => {
      'ip4_addr' => '192.168.101.2'
    },
    'prefix' => {
      'masklen' => '24',
      'ip_addr' => {
        'ip4_addr' => '192.168.101.0'
```

```
      }
    }
  },
  'protocol' => 'BGP',
  'hop_dst' => {
    'type' => 'AS Border Router',
    'ipaddr' => {
      'ip4_addr' => '24.0.0.2'
    }
  },
  'area/AS' => 'lab.BGP/AS65522',
  'lookups' => [
    {
      'protocol' => 'OSPF',
      'metric' => '0',
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '192.168.127.0'
        }
      }
    }
  ],
  'prefix' => {
    'masklen' => '24',
    'ip_addr' => {
      'ip4_addr' => '57.57.57.0'
    }
  }
},
{
  'hop_src' => {
    'type' => 'AS Border Router',
    'ipaddr' => {
      'ip4_addr' => '24.0.0.2'
    }
  },
  'protocol' => 'BGP',
  'hop_dst' => {
    'type' => 'LAN Pseudo-Node',
    'ipaddr' => {
      'ip4_addr' => '192.168.103.2'
```

```
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '192.168.103.0'
        }
      }
    },
    'area/AS' => 'lab.BGP/AS65522',
    'lookups' => [
      {
        'protocol' => 'OSPF',
        'metric' => '1',
        'prefix' => {
          'masklen' => '24',
          'ip_addr' => {
            'ip4_addr' => '192.168.127.0'
          }
        }
      }
    ],
    'prefix' => {
      'masklen' => '24',
      'ip_addr' => {
        'ip4_addr' => '57.57.57.0'
      }
    }
  },
  {
    'hop_src' => {
      'type' => 'LAN Pseudo-Node',
      'ipaddr' => {
        'ip4_addr' => '192.168.103.2'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '192.168.103.0'
        }
      }
    },
    'protocol' => 'BGP',
```

```
                'hop_dst' => {
                  'type' => 'AreaBR, ASBR',
                  'ipaddr' => {
                    'ip4_addr' => '24.0.0.11'
                  }
                },
                'area/AS' => 'lab.BGP/AS65522',
                'lookups' => [
                  {
                    'protocol' => 'OSPF',
                    'metric' => '0',
                    'prefix' => {
                      'masklen' => '24',
                      'ip_addr' => {
                        'ip4_addr' => '192.168.127.0'
                      }
                    }
                  }
                ],
                'prefix' => {
                  'masklen' => '24',
                  'ip_addr' => {
                    'ip4_addr' => '57.57.57.0'
                  }
                }
              }
            ],
            'path' => 'Path 1',
            'num_hops' => '4'
          }
        ]
      }
    ]
}
```

Sample Output (for IPv6) :

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
```

```
'numReturnedEntries' => '1',
'network_name' => 'Dynamips.Lab1',
'network_time' => '20090203T01:18:21',
'report_time' => '20090210T16:31:09',
'totalEntries' => '1',
'result' => [
  {
    'path_dst' => {
      'masklen' => '64',
      'ip_addr' => {
        'ip6_addr' => '2002:cccc::'
      }
    },
    'path_cost' => '40',
    'path_src' => {
      'overloaded' => 'false',
      'sysid' => '49.0003.1760.1600.1031.00',
      'name' => 'r31',
      'type' => 'L2 Internal Router',
      'protoType' => 'IPv4 + IPv6',
      'ipaddr' => {
        'ip4_addr' => '172.16.1.31'
      },
      'ip6addr' => {
        'ip6_addr' => '2001:a331::31'
      }
    },
    'paths' => [
      {
        'cost' => '40',
        'path_hops' => [
          {
            'hop_src' => {
              'overloaded' => 'false',
              'sysid' => '49.0003.1760.1600.1031.00',
              'name' => 'r31',
              'type' => 'L2 Internal Router',
              'protoType' => 'IPv4 + IPv6',
              'ipaddr' => {
                'ip4_addr' => '172.16.1.31'
              },
              'ip6addr' => {
```

```
          'ip6_addr' => '2001:a331::31'
        }
      },
      'protocol' => 'ISIS',
      'hop_dst' => {
        'overloaded' => 'false',
        'sysid' => '1760.1600.1031.02',
        'name' => 'r31.02',
        'type' => 'LAN Pseudo-Node',
        'protoType' => 'IPv4',
        'ipaddr' => {
          'ip4_addr' => '10.3.0.0'
        }
      },
      'area/AS' => 'Dynamips.Lab1.ISIS/Level2',
      'metric' => '10',
      'prefix' => {
        'masklen' => '64',
        'ip_addr' => {
          'ip6_addr' => '2002:cccc::'
        }
      }
    },
    {
      'hop_src' => {
        'overloaded' => 'false',
        'sysid' => '1760.1600.1031.02',
        'name' => 'r31.02',
        'type' => 'LAN Pseudo-Node',
        'protoType' => 'IPv4',
        'ipaddr' => {
          'ip4_addr' => '10.3.0.0'
        }
      },
      'protocol' => 'ISIS',
      'hop_dst' => {
        'overloaded' => 'false',
        'sysid' => '49.0001.1760.1600.1003.00',
        'name' => 'r3',
        'type' => 'L2 Internal Router',
        'protoType' => 'IPv4 + IPv6',
        'ipaddr' => {
```

```
      'ip4_addr' => '172.16.1.3'
    },
    'ip6addr' => {
      'ip6_addr' => '2001:bb03::3'
    }
  },
  'area/AS' => 'Dynamips.Lab1.ISIS/Level2',
  'metric' => '0',
  'prefix' => {
    'masklen' => '64',
    'ip_addr' => {
      'ip6_addr' => '2002:cccc::'
    }
  }
},


.................
.................
.................
.................


{
  'hop_src' => {
    'overloaded' => 'false',
    'sysid' => '49.0002.1760.1600.1021.00',
    'name' => 'r21',
    'type' => 'L2 Internal Router',
    'protoType' => 'IPv4 + IPv6',
    'ipaddr' => {
      'ip4_addr' => '172.16.1.21'
    },
    'ip6addr' => {
      'ip6_addr' => '2001:a221::21'
    }
  },
  'protocol' => 'ISIS',
  'hop_dst' => {
    'overloaded' => 'false',
    'sysid' => '49.0002.1760.1600.1021.00',
    'name' => 'r21',
```

```
                'type' => 'L2 Internal Router',
                'protoType' => 'IPv4 + IPv6',
                'ipaddr' => {
                  'ip4_addr' => '172.16.1.21'
                },
                'ip6addr' => {
                  'ip6_addr' => '2001:a221::21'
                }
              },
              'area/AS' => 'Dynamips.Lab1.ISIS/Level2',
              'metric' => '10',
              'prefix' => {
                'masklen' => '64',
                'ip_addr' => {
                  'ip6_addr' => '2002:cccc::'
                }
              }
            }
          ],
          'path' => 'Path 1',
          'num_hops' => '6'
        }
      ]
    }
  ]
}
```

Example (OSI Network)

```
#!/usr/bin/perl
#*
#*

if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2]) ||
!defined($ARGV[3])) {
    printf "usage: RouteAnalyzer.api_mp_list_osi_paths ip database src
dest\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $srcaddr = $ARGV[2];
my $dstaddr = $ARGV[3];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("01 Feb 2007 16:40:21 PST");

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_list_paths',
                RPC::XML::RPC_STRING($password),
                RPC::XML::RPC_STRING($database),
                new RPC::XML::struct(ip_addr =>
                        new RPC::XML::struct(osi_addr => "$srcaddr"),  ##
srcaddr = 0000.0001.0000.00
                        masklen => 0),    # mask len is not used
                new RPC::XML::struct(ip_addr =>
                        new RPC::XML::struct(osi_addr => "$dstaddr"),  ##
dstaddr = 27.0001.0000.0008.0000.00
                        masklen => 0),   #masklen is not used.
                RPC::XML::datetime_iso8601->new(time2iso8601($t1)))
```

```
                );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output (OSI Network)

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDIHari',
  'report_time' => '20070206T09:15:23',
  'totalEntries' => '1',
  'result' => [
    {
      'path_dst' => '27.0001.0000.0008.0000.00',
      'path_cost' => '10',
      'path_src' => {
        'sysid' => '47.0024.0000.0001.0000.00,27.0001.0000.0001.0000.00',
        'name' => 'Router1',
        'type' => 'L1L2 Router',
        'protoType' => 'OSI'
    },
      'paths' => [
        {
          'cost' => '10',
          'path_hops' => [
            {
              'hop_src' => {
                'sysid' =>
'47.0024.0000.0001.0000.00,27.0001.0000.0001.0000.00',
                'name' => 'Router1',
                'type' => 'L1L2 Router',
                'protoType' => 'OSI'
              },
              'protocol' => 'ISIS',
              'hop_dst' => {
                'sysid' =>
'47.0023.0000.0021.0000.01,47.0024.0000.0021.0000.01,27.0001.0000.0021.00
00.01',
                'type' => 'LAN Pseudo-Node',
                'protoType' => 'OSI'
              },
```

```
              'metric' => '10',
              'prefix' => '27.0001.0000.0008.0000.00'
            },
            {
              'hop_src' => {
                'sysid' =>
'47.0023.0000.0021.0000.01,47.0024.0000.0021.0000.01,27.0001.0000.0021.00
00.01',
                'type' => 'LAN Pseudo-Node',
                'protoType' => 'OSI'
              },
              'protocol' => 'ISIS',
              'hop_dst' => {
                'sysid' => '27.0001.0000.0008.0000.00',
                'name' => 'Router8',
                'type' => 'L1 Internal Router',
                'protoType' => 'OSI'
              },
              'metric' => '0',
              'prefix' => '27.0001.0000.0008.0000.00'
            },
            {
              'hop_src' => {
                'sysid' => '27.0001.0000.0008.0000.00',
                'name' => 'Router8',
                'type' => 'L1 Internal Router',
                'protoType' => 'OSI'
              },
              'protocol' => 'ISIS',
              'hop_dst' => {
                'sysid' => '27.0001.0000.0008.0000.00',
                'name' => 'Router8',
                'type' => 'L1 Internal Router',
                'protoType' => 'OSI'
              },
              'metric' => '0',
              'prefix' => '27.0001.0000.0008.0000.00'
            }
          ],
          'path' => 'Path 1',
          'num_hops' => '3'
        }
```

```
        ]
      }
    ]
}
```

api_mp_open_handle

RPC Call: RC Call: RouterAnalyzer.api_mp_open_handle {password}

This query lists all of the call handles that are still open, along with the types of calls they handle and their creation time.

Input Parameters

- **password** – The password configured for queries.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string (blank for this call)
- network_time: ISO 8601 UTC time
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - handle: int
  - type: string
  - time:
    - seconds: int
    - useconds: int

Example

```
#!/usr/bin/perl
```

```
if(!defined($ARGV[0])) {
    printf "usage: RouteAnalyzer.api_mp_close_handle ip \n";
    exit(0);
}

my $qsip = $ARGV[0];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $openreq = RPC::XML::request->new('RouteAnalyzer.api_mp_open_handle',
                RPC::XML::RPC_STRING($password)
                );

my $openres = $client->send_request($openreq);
if ($openres->is_fault) {print("---XMLRPC FAULT ---"); }
my $overall = $openres->value;

print Dumper($overall);
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => '',
  'network_time' => '20030723T18:12:09',
  'report_time' => '20090402T19:54:57',
```

```
  'totalEntries' => '2',
  'result' => [
    {
      'handle' => '162035008',
      'time' => {
        'seconds' => '1238702046',
        'useconds' => '0'
      },
      'type' => 'api_mp_routes'
    },
    {
      'handle' => '162193072',
      'time' => {
        'seconds' => '1238702093',
        'useconds' => '0'
      },
      'type' => 'api_mp_events'
    }
  ]
}
```

api_mp_osi_routes

RPC Call: RouteAnalyzer**.** api_mp_osi_routes {password} {database name} {time} {filter} {max entries}

This query returns a handle for all routes, including all Prefix Neighbors and ES Neighbors announcements from all routers announcing the Prefix Neighbors and ES Neighbors at the specified time, and meeting the specified criteria.

The query can return a large number of routes in a short amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional [max entries] parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone (for example, 20050725T21:47:35). The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **max entries** – An optional 32-bit integer parameter specifying the maximum number of entries to return in the query.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following structures:
  - topology: topology struct
  - attributes: ISIS attribute struct
  - Prefix Neighbor/ES Neighbor: string
  - router: string
  - state: MP state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_mp_osi_routes ip database [filter] \n";
```

```
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("1 Feb 2007 16:50:00 PST");

# 10K entries default; if -1 entered, RPC implementation will return all
my $num = 2;
$num = $ARGV[3] if ($#ARGV >= 3);

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_osi_routes',
                        RPC::XML::RPC_STRING($password),
                        RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                        RPC::XML::RPC_STRING($filter),
                        RPC::XML::RPC_INT($num))
                        );


foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

Sample Output

```
      {
'vinfo' =>
  'software_version' => '8.0.30-R Traffic Explorer',
  'appliance_version' => '8.0.30-R'
},
'numReturnedEntries' => '2',
'network_name' => 'PDIHari',
'report_time' => '20070206T09:03:48',
'totalEntries' => '66',
'result' => [
  {
    'Prefix Neighbor' => '47.0010.0001',
    'topology' => {
      'fullName' => 'PDIHari.ISIS/Level2',
      'protocol' => 'ISIS'
    },
    'attributes' => {
      'metric' => '0',
      'metricType' => 'Prefix Neighbor Comparable'
    },
    'router' => {
      'sysid' => '47.0024.0000.0001.0000.00,27.0001.0000.0001.0000.00',
      'name' => 'Router1',
      'type' => 'L1L2 Router',
      'protoType' => 'OSI'
    },
    'state' => {
      'inBaseline' => 'false',
      'down' => 'true'
    }
  },
  {
    'Prefix Neighbor' => '47.0010.0001',
    'topology' => {
      'fullName' => 'PDIHari.ISIS/Level2',
      'protocol' => 'ISIS'
    },
    'attributes' => {
      'metric' => '0',
      'metricType' => 'Prefix Neighbor Comparable'
```

```
      },
      'router' => {
        'sysid' =>
'47.0023.0000.0021.0000.00,47.0024.0000.0021.0000.00,27.0001.0000.0021.00
00.00',
        'name' => 'Router21',
        'type' => 'L1L2 Router',
        'protoType' => 'OSI'
      },
      'state' => {
        'inBaseline' => 'false',
        'down' => 'false'
      }
    }
  ]
}
```

api_mp_osi_routes_handle

RPC Call: RouteAnalyzer.api_mp_osi_routes {password} {database name} {time} {filter}

This query returns a handle for all routes, including all Prefix Neighbors and ES Neighbors announcements from all routers announcing the Prefix Neighbors and ES Neighbors at the specified time, and meeting the specified criteria.

The query can return a large number of routes in a short amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional [max entries] parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.

- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone (for example, 20050725T21:47:35). The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: int

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample details.

api_mp_prefixes_multi_origin

RPC Call: RouteAnalyzer.api_mp_prefixes_multi_origin {password} {database name} {time}{threshold} {max entries}

This query returns a list of IPv4 and IPv6 prefixes for the specified network that are originated by more than one router (or Intermediate System in OSI terminology).

Input Parameters

- **password** – The password configured for queries.

- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **threshold —** A threshold is for the minimum number of originations of a prefix in the reports. The minimum number is 2.
- **max entries** – An optional 32-bit integer parameter specifying the maximum number of entries to return in the query.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- prefixes: array of the following:prefixes:
  - prefix or prefix6: string
  - prefix_area: string
  - prefix_type: string
  - router: MP router struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_prefixes_multi_origin ip database
[filter] \n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $thresh = 2;

$thresh = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";


my $t1 = str2time("5 Feb 2007 19:50:00 PST");

# 10K entries default; if -1 entered, RPC implementation will return all
my $num = 10000;
$num = $ARGV[3] if ($#ARGV >= 3);

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_mp_prefixes_multi_origin',
                        RPC::XML::RPC_STRING($password),
                        RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                        RPC::XML::RPC_INT($thresh),
                        RPC::XML::RPC_INT($num))
                        );
```

```
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '42',
  'network_name' => 'Audi.IPv6',
  'network_time' => '20090206T12:36:37',
  'report_time' => '20090206T12:36:37',
  'totalEntries' => 42,
  'result' => [
    {
      'area' => 'Audi.IPv6.ISIS/Level2',
      'type' => 'Internal',
      'prefix' => '10.55.82.0/24'
    },
    {
      'area' => 'Audi.IPv6.ISIS/Level2',
      'type' => 'Internal',
      'prefix' => '10.55.82.0/24',
      'router' => {
        'overloaded' => 'false',
        'sysid' => '49.1111.1960.1600.1001.00',
        'name' => 'one',
        'type' => 'L2 Internal Router',
        'protoType' => 'IPv4 + IPv6',
        'ipaddr' => '175.16.1.1',
        'ip6addr' => '2001:cc04::4'
      }
    },
    {
      'area' => 'Audi.IPv6.ISIS/Level2',
      'type' => 'Internal',
      'prefix' => '10.55.82.0/24',
      'router' => {
        'overloaded' => 'false',
        'sysid' => '49.0001.1760.1600.1001.00',
        'name' => 'r1',
        'type' => 'L2 Internal Router',
```

```
          'protoType' => 'IPv4 + IPv6',
          'ipaddr' => '172.16.1.4',
          'ip6addr' => '2001:bb04::4'
        }
      },
      {
        'prefix6' => '2001:cc13::/64',
        'area' => 'Audi.IPv6.ISIS/Level2',
        'type' => 'Internal'
      },
      {
        'prefix6' => '2001:cc13::/64',
        'area' => 'Audi.IPv6.ISIS/Level2',
        'type' => 'Internal',
        'router' => {
          'overloaded' => 'false',
          'sysid' => '49.1111.1960.1600.1001.00',
          'name' => 'one',
          'type' => 'L2 Internal Router',
          'protoType' => 'IPv4 + IPv6',
          'ipaddr' => '175.16.1.1',
          'ip6addr' => '2001:cc04::4'
        }
      },
      {
        'prefix6' => '2001:cc13::/64',
        'area' => 'Audi.IPv6.ISIS/Level2',
        'type' => 'Internal',
        'router' => {
          'overloaded' => 'false',
          'sysid' => '49.1111.1960.1600.1003.00',
          'name' => 'three',
          'type' => 'L2 Internal Router',
          'protoType' => 'IPv4 + IPv6',
          'ipaddr' => '175.16.1.3',
          'ip6addr' => '2001:cc03::3'
        }
      },
.....
```

api_mp_prefixes_multi_origin_handle

RPC Call: RouteAnalyzer.api_mp_prefixes_multi_origin_handle {password} {database name} {time} {threshold}

This query returns a list of prefixes for the specified network that are originated by more than one router (or Intermediate System in OSI terminology).

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **threshold —** A threshold is for the minimum number of originations of a prefix in the reports. The minimum number is 2.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- prefixes: array of the following:
  - prefix or prefix6: string
  - prefix_area: string
  - prefix_type: string
  - router: MP router struct

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample output details.

api_mp_routers

RPC Call: RouteAnalyzer.api_mp_routers {password} {database name} {time} {filter}{pseudonodes}

This query lists all routers present in the multi-protocol network at the specified time. The results may be filtered to select only the routers running a particular protocol, for example.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **pseudonodes** – An optional string parameter set to "true" or "1" to include pseudonodes in addition to routers in the call output.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - topology: topology struct
  - numPrefixes: int
  - numIPv6Prefixes: int

— router: MP router struct

— state: MP state struct (without baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_routers ip database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Feb 2005 15:50:22 PST");

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_routers',
             RPC::XML::RPC_STRING($password),
             RPC::XML::RPC_STRING($database),
             RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
             RPC::XML::RPC_STRING($filter)));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
 'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '91',
  'network_name' => 'PacketDesignIPv6.AS64600',
  'network_time' => '20090116T09:28:00',
  'report_time' => '20090123T12:42:46',
  'totalEntries' => '91',
  'result' => [
    {
      'numIPv6Prefixes' => '7',
      'topology' => {
        'fullName' => 'PacketDesignIPv6.AS64600.ISIS/49.0100',
        'protocol' => 'ISIS'
      },
      'numPrefixes' => '7',
      'router' => {
        'overloaded' => 'false',
        'sysid' => '49.0100.1760.1600.1001.00',
        'name' => 'R1',
        'type' => 'L1L2 Router',
        'protoType' => 'IPv4 + IPv6',
        'ipaddr' => '172.16.1.1',
        'ip6addr' => '2008:bb01::1'
      },
      'state' => {
        'down' => 'false'
      }
    },
    {
      'numIPv6Prefixes' => '7',
      'topology' => {
        'fullName' => 'PacketDesignIPv6.AS64600.bgp.ip6.BGP/AS64600/IPv6',
        'protocol' => 'BGP'
      },
      'numPrefixes' => '0',
      'router' => {
        'name' => 'R1',
```

```
          'type' => 'IBGP Peer',
          'ipaddr' => '172.16.1.1'
        },
        'state' => {
          'down' => 'false'
        }
      },
      {
'topology' => {
          'fullName' => 'PacketDesignIPv6.AS64600.Static/snmp',
          'protocol' => 'Static'
        },
        'numPrefixes' => '5',
        'router' => {
          'name' => 'R34',
          'model' => '3600',
          'type' => 'Static',
          'softwareVersion' => '12.4(10a)',
          'ipaddr' => '172.16.1.34'
        },
        'state' => {
          'down' => 'false'
        }
      },
......
```

api_mp_routers_consolidated

RPC Call: RouteAnalyzer.api_mp_routers_consolidated{password} {database name} {time} {filter}

This query lists all the mp-level nodes present in the multi-protocol network at the specified time. For each mp-level node, the following information is provided: Sysid, Number of rt nodes, and Array of rt nodes. For each rt-node, the following information is provided: router id, protocol, router type, number of interfaces for this rt node, array of interfaces ips for this router.

Input Parameters

password—The password configured for the queries.

database name—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

time—A time-specified in ISO 8601 in the UTC time zone, such as 20050725T21:47:35. The query results will be calculated based on the network state at the specified time.

filter—A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- result: array of the following:
  - mpID: string
  - rtNodeCount: int
- array of rtNodes struct which is as follows:
  - id: string
  - type: string
  - intfCount: int
  - Array of interfaces ips: array of strings.

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_routers_consolidated ip database
[filter]\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("24 Jun 2008 11:55:17 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_mp_routers_consolidated',
RPC::XML::RPC_STRING($password),
RPC::XML::RPC_STRING($database),
RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
RPC::XML::RPC_STRING($filter) )
);

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
print Dumper($value1);
```

```
}
```

Sample Output

```
'vinfo' => {
  'software_version' => '8.0.30-R Traffic Explorer',
  'appliance_version' => '8.0.30-R'
},
'numReturnedEntries' => '17',
'network_name' => 'Lab1',
'report_time' => '20080624T08:24:07',
'totalEntries' => '17',
'result' => [
  {
    'mpNode' => {
      'rtNodeCount' => '1',
      'rtNodes' => [
        {
          'proto' => 'ISIS',
          'intfCount' => '0',
          'type' => 'L1 Internal Router',
          'id' => '255.255.255.255'
        }
      ],
      'mpID' => '0x0000000600000000'
    }
  },
  {
    'mpNode' => {
      'rtNodeCount' => '1',
      'rtNodes' => [
        {
          'proto' => 'OSPF',
          'intfCount' => '3',
          'type' => 'AS Border Router',
          'id' => '24.0.0.12',
          'intfs' => [
            {
              'ip' => '10.12.113.1'
            },
            {
              'ip' => '102.0.1.1'
```

```
                },
                {
                  'ip' => '192.168.103.12'
                }
              ]
            }
          ],
          'mpID' => '0x1800000C00000001'
        }
      },
      ....
    ]
  }
},
{
'ip' => '192.168.103.12'
}
]
}
],
'mpID' => '0x1800000C00000001'
}
},
```

api_mp_routers_consolidated_handle

RPC Call: RouteAnalyzer.api_mp_routers_consolidated{password} {database name} {time} {filter}

This query returns a handle for all the mp-level nodes present in the multi-protocol network at the specified time. For each mp-level node, the following information is provided: Sysid, Number of rt nodes, and Array of rt nodes. For each rt-node, the following information is provided: router id, protocol, router type, number of interfaces for this rt node, array of interfaces ips for this router.

Input Parameters

password—The password configured for the queries.

database name—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

time—A time-specified in ISO 8601 in the UTC time zone, such as 20050725T21:47:35. The query results will be calculated based on the network state at the specified time.

filter—A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- result: array of the following:
  - mpID: string
  - rtNodeCount: int
- array of rtNodes struct which is as follows:
  - id: string
  - type: string
  - intfCount: int
  - Array of interfaces ips: array of strings.

Example and Sample Output

See Using Re-Entrant Queries on page 39 for example and sample details.

api_mp_routes

RPC Call: RouteAnalyzer.api_mp_routes {password} {database name} {time} {filter} {max entries}

This query lists all routes including all prefix announcements from all routers announcing the prefixes, at the specified time and meeting the specified filter criteria.

The query can return a large number of BGP routes in a small amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional {max entries} parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **max entries** – An optional 32-bit integer parameter specifying the maximum number of entries to return in the query.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - topology: topology struct

- attributes: LS attribute struct (if it is LS)
- attributes: EIGRP attribute struct (if it is EIGRP)
- attributes: string (if other IGP)
- attributes: Static attribute struct (if it is Static)
- attributes: BGP: attribute struct (if it is BGP)
- attributes: string (if other)
- prefix: string
- router: MP router struct
- state: MP state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_routes ip database filter\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = time;

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_mp_routes',
             RPC::XML::RPC_STRING($password),
```

```
                RPC::XML::RPC_STRING($database),
                RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                RPC::XML::RPC_STRING($filter), 150));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output


```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1042',
  'network_name' => 'PDI',
  'network_time' => '20090305T09:49:57',
  'report_time' => '20090305T09:49:58',
  'totalEntries' => '1042',
  'result' => [
     {
      'topology' => {
        'fullName' => 'PDI.OSPF/0.0.0.2',
        'protocol' => 'OSPF'
      },
      'attributes' => {
        'metric' => '11113',
        'metricType' => 'Area External'
      },
      'prefix' => '10.101.244.4/30',
      'router' => {
        'name' => 'Router26.lab.packetdesign.com',
        'type' => 'AreaBR',
        'ipaddr' => '10.130.1.26'
      },
      'state' => {
        'inBaseline' => 'false',
        'down' => 'false'
      }
    },
    {
      'topology' => {
        'fullName' => 'PDI.BGP/AS65464',
        'protocol' => 'BGP'
      },
      'attributes' => {
        'origin' => 'IGP',
        'localPref' => '100',
        'nextHop' => '10.64.16.11',
```

```
      'asPath' => '',
      'med' => '30'
    },
    'prefix' => '11.11.7.0/24',
    'router' => {
      'name' => 'DC-CORE1-ROUTER3.lab.packetdesign.com',
      'type' => 'IBGP Peer',
      'ipaddr' => '10.120.1.3'
    },
    'state' => {
      'inBaseline' => 'true',
      'down' => 'false'
    }
  },
  {
    'topology' => {
      'fullName' => 'PDI.ISIS/Level2',
      'protocol' => 'ISIS'
    },
    'attributes' => {
      'metric' => '0',
      'metricType' => 'internal'
    },
    'prefix' => '11.11.7.0/24',
    'router' => {
      'overloaded' => 'false',
      'sysid' => '47.0001.0000.0000.000A.00',
      'name' => 'SF-PE1-ROUTER6',
      'type' => 'L2 Internal Router',
      'protoType' => 'IPv4 + IPv6',
      'ipaddr' => '10.120.1.6',
      'ip6addr' => '2009:6666::a78:106'
    },
    'state' => {
      'inBaseline' => 'false',
      'down' => 'false'
    }
  },
  {
    'topology' => {
      'fullName' => 'PDI.Static/snmp',
      'protocol' => 'Static'
```

```
    },
    'attributes' => {
      'nextHops' => [
        {
          'nextHop' => '2'
        }
      ]
    },
    'prefix' => '169.254.0.0/32',
    'router' => {
      'name' => 'router212.example.com',
      'model' => '',
      'type' => 'Static',
      'softwareVersion' => '',
      'ipaddr' => '10.71.2.212'
    },
    'state' => {
      'inBaseline' => 'false',
      'down' => 'false'
    }
  }
  .....
]
```

api_mp_routes_handle

RPC Call: RouteAnalyzer.api_mp_routes {password} {database name} {time} {filter}

This query returns a handle for all routes, including all prefix announcements from all routers announcing the prefixes at the specified time, and meeting the specified filter criteria.

The query can return a large number of BGP routes in a small amount of time. You can keep the number of routes manageable by refining your filter. You may also have to expand XML RPC client timeouts to accommodate the amount of time required for the query to acquire all the routes. Alternatively, you can supply the optional {max entries} parameter to limit the number of entries returned.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: int

Example and Sample Output

See Using Re-Entrant Queries `on page 39` for example and sample output details.

api_mp_vpn_connections

RPC Call: RouteAnalyzer.api_mp_vpn_connections {password} {database name} {time} {filter} {staticPrefixes}

This query lists all the VPN connections to sites in an MPLS WAN network. These connections are configured as described in Setting Up the VPN Connection Configuration in the User's Guide.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.
- **staticPrefixes** - An optional boolean flag to control whether the result should include a listing of the prefix groups configured for sites connected by static routes. These prefix groups are configured as described in "Setting Up Static VPN Connections" in the *HP Route Analytics Management System User's Guide*.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- network_time: ISO 8601 UTC time
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of structs containing the following:
  - site: string
  - ceRouter: MP router struct
  - peRouter: MP router struct
  - protocol: string
- as: struct (present if protocol is BGP)
  - name: string
  - number: int

- state: string
- vpn: string
- prefixGroup: string (present if protocol is Static and staticPrefixes is true)
- announcedPrefixes: array of structs containing the following (present if protocol is Static and staticPrefixes is true)
  - prefix: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_mp_vpn_connection ip database [filter]
[staticAnnouncedPrefixes]\n";
    exit(0);
}

my $rexip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = str2time("22 Sep 2010 20:51:27 PDT");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_mp_vpn_connections',
                RPC::XML::RPC_STRING($password),
                RPC::XML::RPC_STRING($database),
```

```
                RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                RPC::XML::RPC_STRING($filter),
RPC::XML::RPC_BOOLEAN($ARGV[3]))
                );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '9.0.47-E Traffic Explorer',
    'appliance_version' => '9.0.47'
  },
  'numReturnedEntries' => '3',
  'network_name' => 'Enterprise',
  'network_time' => '2010-09-23T03:51:27',
  'report_time' => '2010-09-24T02:24:33',
  'totalEntries' => '3',
  'result' => [
    {
      'protocol' => 'Collector',
      'announcedPrefixes' => [
        {
          'prefix' => '10.75.1.0/24'
        },
        {
          'prefix' => '10.75.2.0/24'
        },
        {
          'prefix' => '10.105.200.0/30'
        },
        {
          'prefix' => '10.134.1.60/32'
        }
      ],
```

```
    'prefixGroup' => 'CA_Static_Site3',
    'state' => 'Configured',
    'site' => 'Enterprise.CA',
    'ceRouter' => {
      'mpname' => 'SITE3-CE-RTR60',
      'name' => 'SITE3-CE-RTR60 ',
      'model' => 'Cisco',
      'type' => 'Device',
      'softwareVersion' => 'IOS',
      'ipaddr' => '10.134.1.60',
      'serialNumber' => '26788269'
    },
    'peRouter' => {
      'mpname' => '10.75.1.0/24',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => '10.75.1.0',
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.75.1.0'
        }
      }
    },
    'vpn' => 'AS_65464'
  },
  {
    'protocol' => 'BGP',
    'site' => 'Enterprise.NY',
    'as' => {
      'number' => '65464',
      'name' => 'private'
    },
    'ceRouter' => {
      'mpname' => 'OSPF-SITE-CE-RTR21',
      'name' => 'OSPF-SITE-CE-RTR21 ',
      'type' => 'IBGP Peer',
      'ipaddr' => '10.130.1.21'
    },
    'peRouter' => {
      'mpname' => '10.71.1.7(AS65464)',
```

```
          'type' => 'EBGP NextHop',
          'ipaddr' => '10.71.1.7'
        },
        'state' => 'Configured',
        'vpn' => 'AS_65464'
      },
      {
        'protocol' => 'BGP',
        'site' => 'Enterprise.Texas',
        'as' => {
          'number' => '65464',
          'name' => 'private'
        },
        'ceRouter' => {
          'mpname' => 'EIGRP-SITE-CE-RTR20',
          'name' => 'EIGRP-SITE-CE-RTR20',
          'type' => 'IBGP Peer',
          'ipaddr' => '10.132.1.20'
        },
        'peRouter' => {
          'mpname' => '10.70.1.6(AS65464)',
          'type' => 'EBGP NextHop',
          'ipaddr' => '10.70.1.6'
        },
        'state' => 'Configured',
        'vpn' => 'AS_65464'
      }
    ]
}
```

api_prefix_list_multi_orig

RPC Call: RouteAnalyzer.api_prefix_list_multi_orig {password} {database name} {time}

This query returns a list of prefixes for the specified network that are originated by more than one router.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

Structure of Output

- vinfo: version struct
- network_name: string
- report time: ISO 8601 UTC time
- prefixes: array of the following:
  - routers: array of router structs
  - prefix_type: string
  - prefix_area: string
  - prefix: prefix struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_prefix_list_multi_orig ip database\n";
     exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
```

```
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = 1058927123;

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_prefix_list_multi_orig',
           RPC::XML::RPC_STRING($password),
           RPC::XML::RPC_STRING($database),
           RPC::XML::datetime_iso8601->new(time2iso8601($t1)) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print ("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
print (STDERR join "\n", Dumper($value1) );

}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'network_name' => 'pd353',
  'report_time' => '20051028T00:45:15',
  'prefixes' => [
    {
      'routers' => [
        {
          'nodeType' => 'ASBR',
          'ip_addr' => {
            'ip4_addr' => '192.168.120.120'
          },
          'nodeState' => 'DOWN',
          'nodeProto' => 'Static',
          'name' => 'Router16',
          'nodeArea' => 'pd353.Left.EIGRP/AS1',
```

```
            'maskLen' => '32',
            'systemID' => '192.168.120.120'
          },
          {
            'nodeType' => 'ASBR',
            'ip_addr' => {
              'ip4_addr' => '192.168.122.122'
            },
            'nodeState' => 'DOWN',
            'nodeProto' => 'Static',
            'name' => 'Router26',
            'nodeArea' => 'pd353.Left.EIGRP/AS1',
            'maskLen' => '32',
            'systemID' => '192.168.122.122'
          },
          {
            'nodeType' => 'Internal',
            'ip_addr' => {
              'ip4_addr' => '192.168.220.20'
            },
            'nodeState' => 'DOWN',
            'nodeProto' => 'Static',
            'name' => 'Router20',
            'nodeArea' => 'pd353.Left.EIGRP/AS1',
            'maskLen' => '32',
            'systemID' => '192.168.220.20'
          },
        ],
        'prefix_type' => 'Static',
        'prefix_area' => 'AllStaticRoutes.Static',
        'prefix' => {
          'masklen' => '0',
          'ip_addr' => {
            'ip4_addr' => '0.0.0.0'
          }
        }
      }
      ....
    ]
  }
```

api_resource_status

RPC Call: RouteAnalyzer.api_resource_status {password}

This query lists the current status of the memory, disk, and swap space on the appliance. This displays the used, free, and total amounts, along with the percentage of user, system, idle and other CPU utilization.

Input Parameters

- **password** – The password configured for queries.

Structure of Output

- vinfo: version struct
- resources:
  - memory:
    - free: int
    - used: int
    - total: int
    - pct: double (percentage of memory currently used)
  - disk:
    - free: int
    - used: int
    - total: int
    - pct: double (percentage of memory currently used)
  - swap:
    - free: int
    - used: int
    - total: int
    - pct: double (percentage of memory currently used)
  - cpu:
    - user: double (percentage)

- system: double (percentage)
- idle: double (percentage)
- other: double (percentage; “other” consists of any remaining CPU usage such as niced processes, I/O waiting, servicing hardware and software interrupts, etc.)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0])) {
    printf "usage: RouteAnalyzer.api_resource_status ip\n";
    exit(0);
}

my $qsip = $ARGV[0];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_resource_status',
                RPC::XML::RPC_STRING($password)
));

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'resources' => {
    'memory' => {
      'pct' => '71.70806685821979',
      'free' => '293032',
      'used' => '742712',
      'total' => '1035744'
    },
    'disk' => {
      'pct' => '79.65773029037497',
      'free' => '5195956',
      'used' => '27265044',
      'total' => '34227744'

    },
    'cpu' => {
```

api_router_summarizable

RPC Call: RouteAnalyzer.api_router_summarizable {password} {database name} {time}

This query returns a list of routers that, at the specified time, are advertising multiple prefixes that could be summarized as a single prefix. For each such router, the appliance provides a list of potential summary prefixes with their component prefixes (both IPv4 and IPv6 prefixes). Prefixes that are internal (native to the IGP) and those that are external (imported from another routing protocol) are considered separately.

Input Parameters

- **password** – The password configured for queries.

- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

Structure of Output

- vinfo: version structs
- report_time: ISO 8601 UTC time
- network_name: string
- routers: array of the following:
  - router: router struct
  - summarizable_prefixes: array of the following:
    - summary: prefix IP struct
    - contributors: array of prefix IP structs

Example

```
#!/usr/bin/perl
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $t1 = time2iso8601(time);
my $request = RPC::XML::request->new(
      'RouteAnalyzer.api_router_summarizable',
       RPC::XML::RPC_STRING( 'admin' ),   //password
       RPC::XML::RPC_STRING( 'CorpNet' ),  //database name
       RPC::XML::datetime_iso8601->new($t1)
       );
my $client = new RPC::XML::Client 'http://hostname:2000/RPC2';
my $result = $client->send_request($request);
if ($result->is_fault) { print("--- XMLRPC FAULT ---"); }
print(STDERR join "\n", "--- XMLRPC RESULT ---", Dumper($result->value),
'');
```

Sample Output

```
--- XMLRPC RESULT ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer'
    'appliance_version' => '8.0.30-R'
  },
  'report_time' => '20030303T21:09:29',
  'network_name' => 'CorpNet',
  'routers' => [
    {
      'router' => {
        'nodeProto' => 'ospf',
        'ip_addr' => {
            'ip4_addr' => '192.168.140.140'
         },
        'nodeType' => 'AreaBR',
        'name' => '',
        'systemID' => '004001001012:00'
      },
      'summarizable_prefixes' => [
        {
          'summary' => {
            'ip_addr' => {
                'ip4_addr' => '192.168.150.150'
            },
            'masklen' => '31'
          }, // end summary
          'contributors' => [
            {
              'ip_addr' => {
                   'ip4_addr' => '192.168.150.150'
              },
              'masklen' => '32'
            },
            {
              'ip_addr' => {
                    'ip4_addr' => '192.168.150.151'
              },
              'masklen' => '32'
            }
```

```
    ] // end contributors
  },
  {'summary' ...  'contributors' },
  {'summary' ...  'contributors' }
    ] // end summarizable_prefixes
  }, // end first router
 {'router' => {...}, 'summarizable_prefixes' => [...]},
  {'router' => {...}, 'summarizable_prefixes' => [...]}
 ] // end routers
}
```

api_system_health

RPC Call: RouteAnalyzer.api_system_health {password}

This query lists the health of all the RAMS and RAMS Traffic systems in the network, including the recording and writing status of each configured recording process and its databases, along with the location of core files existing on each system. Non-master units can only look at their local unit, while master units can look at the status of each of their clients.

Clients are required to have the same query password as the Master.

If you are calling this query from a Master unit, the call forces the output to be non-brief, even if the call api_conn_brief_xml was used during the connection. Because all clients associated with the Master are queried and the results are combined, the overall output cannot be brief if the client outputs aren't brief.

Input Parameters

- **password** – The password configured for queries.

Structure of Output

- vinfo: version struct
- units: array of the following:

- reachable: int
- ipaddr: string
- processes: array of the following:
  - globaldbname: string
  - running: int
  - process: string
  - dbs: array of the following:
    - dbname: string
    - messages: array of the following:
      - msg: string
      - last_write_time: ISO 8601 UTC time
- cores: array of the following:
  - time: ISO 8601 UTC time
  - file: string
  - size: int
  - process: string

Example

```
if(!defined($ARGV[0])) {
    printf "usage: RouteAnalyzer.api_system_health ip\n";
    exit(0);
}

my $qsip = $ARGV[0];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
```

```
push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_system_health',
                RPC::XML::RPC_STRING($password)
));

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'units' => [
    {
      'reachable' => '0',
      'processes' => [],
      'ipaddr' => '192.168.3.44'
    },
    {
      'reachable' => '1',
      'processes' => [
        {
          'label' => 'Traffic1',
          'running' => '0',
          'dbs' => [],
          'process' => 'Flow Collector'
        }
      ],
      'ipaddr' => '192.168.3.126'
    },
    {
      'reachable' => '1',
      'processes' => [
```

```
        {
          'globaldbname' => 'JustBGP',
          'running' => '1',
          'dbs' => [
            {
              'messages' => [
                {
                  'msg' => 'BGP Recorder is running'
                },
                {
                  'msg' => '1 of 1 peers established'
                }
              ],
              'dbname' => 'JustBGP.BGP/AS65522',
              'last_write_time' => '20061208T21:42:21'
            }
          ],
          'process' => 'BGP Recorder'
        }
      ],
      'ipaddr' => '192.168.3.144'
    }
  ]
}
```

api_unit_health

RPC Call: RouteAnalyzer.api_unit_health {password}

This query lists the health of the specified unit, including the recording and writing status of each configured recording process and its databases, along with the location of the core files existing on the system.

Input Parameters

- **password** – The password configured for queries.

Structure of Output

- reachable: int
- ipaddr: string
  - — processes: array of the following:
  - globaldbname: string
  - running: int
  - process: string
  - dbs: array of the following:
    - — dbname: string
    - — messages: array of the following:
      - — msg: string
      - — last_write_time: ISO 8601 UTC time
  - — cores: array of the following:
  - time: ISO 8601 UTC time
  - file: string
  - size: int
  - process: string

Example

```
if(!defined($ARGV[0])) {
    printf "usage: RouteAnalyzer.api_unit_health ip\n";
```

```
    exit(0);
}

my $qsip = $ARGV[0];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_unit_health',
                RPC::XML::RPC_STRING($password)
));

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}

}
```

Sample Output

```
{
}
  ]
{
  'reachable' => 'true',
  'processes' => [
    {
      'globaldbname' => '',
      'running' => 'true',
      'dbs' => [],
```

```
      'process' => 'Prefix Feeder'
    },
    {
      'globaldbname' => '',
      'running' => 'true',
      'dbs' => [],
      'process' => 'Query Server'
    },
    {
      'globaldbname' => '',
      'running' => 'true',
      'dbs' => [],
      'process' => 'Route Analyzer'
    }
    {
      'globaldbname' => 'JustBGP',
      'running' => 'true',
      'dbs' => [
        {
          'messages' => [
            {
              'msg' => 'BGP Recorder is running'
            },
            {
              'msg' => '1 of 1 peers established'
            }
          ],
          'dbname' => 'JustBGP.BGP/AS65522',
          'last_write_time' => '20061208T21:42:21'
        }
      ],
      'process' => 'BGP Recorder'
  ],
  'ipaddr' => '192.168.1.216',
  'cores' => []
}
```

api_unload_topology

RPC Call: RouteAnalyzer.api_unload_topology {password}

This query removes all applied edits from the loaded topology and unlocks the topology to allow other queries to change the topology and/or time.

Input Parameters

- **password** – The password configured for queries.

Structure of Output

This query is always successful and returns just a message indicating success. The structure of the output is as follows:

- vinfo: version struct
- result: string containing the message

Sample Output

```
{
  'vinfo' => {
     'software_version' => '9.0.30-R Traffic Explorer',
     'appliance_version' => '9.0.30-R'
  },
  'result' => 'Successfully removed edits from topology.',
}
```

api_vpn_cust_rt_list

RPC Call: RouteAnalyzer.api_vpn_cust_rt_list {password} {database name} {operation} {customer name} {route target}

This query returns a list of all VPN customer name to route target (RT) mappings for the specified database. When issued with the `get` operation, no change is made to the list of mappings.
This query also supports additional operations (`add`, `del`, `reset`) to modify the list of mappings, as specified below, in addition to returning the list.

Input Parameters

- **password** – The password configured for queries.
- **database name** – May be an administrative domain, such as CorpNet, which selects the VPN database included in the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **operation** – The specific operation to be performed. This is indicated by a string that can have the value '`get`' to return the list of mappings, '`add`' to add a VPN customer, '`del`' to delete a VPN customer, and '`reset`' to delete all the mappings.
- **customer name** – The empty string for the `get` and `reset` operations; the name of the VPN customer for the `add` and `del` operations.
- **route target** – The empty string for the `get` and `reset` operations; the name of the route target for the `add` and `del` operations.

Structure of Output

- vinfo: version struct
- network_name: string
- vpn_cust_rts: array of the following:
  - name: string
  - rt: string

Example

```
#!/usr/bin/perl
if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_cust_rt_list ip database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_vpn_cust_rt_list',
                                    RPC::XML::RPC_STRING($password),
                                    RPC::XML::RPC_STRING($database),
                                    RPC::XML::RPC_STRING('get'),
                                    RPC::XML::RPC_STRING(''),
                                    RPC::XML::RPC_STRING('')
                                ));

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
```

```
    'appliance_version' => '8.0.30-R'
  },
  'network_name' => 'CorpNet',
  'vpn_cust_rts' => [
    {
      'name' => 'Customer1',
      'rt' => 'RT:65535:101'
    },
    {
      'name' => 'Customer2',
      'rt' => 'RT:65533:101'
    }
  ]
}
```

api_vpn_customer_pe_participation

RPC Call: RouteAnalyzer.api_vpn_customer_pe_participation {password} {database name} {time} {filter}

This query returns statistics of participating PEs for each VPN customer.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: 50
- network_name: string
- report time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - customer: string
  - numActivePEs: int
  - deviation: int
  - numNewPEs: int
  - numDownPEs: int
  - definition: string
  - numBaselinePEs: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_customer_pe_participation ip
database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Feb 2005 15:50:22 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_pe_participation',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '50',
  'network_name' => 'VOD',
  'report_time' => '20051115T19:19:00',
  'totalEntries' => '50',
  'result' => [
    {
      'customer' => 'Cust747',
      'numActivePEs' => '0',
      'deviation' => '100',
      'numNewPEs' => '0',
      'numDownPEs' => '0',
      'definition' => 'RT:600:1',
      'numBaselinePEs' => '0'
    }
    ....
  ]
}
```

api_vpn_customer_pe_list

RPC Call: RouteAnalyzer.api_vpn_customer_privacy {password} {database name} {time} {customer name} {filter}

This query returns the list of participating PEs for the specified VPN customer.

Input Parameters

- **password** – The password configured for queries.

- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **customer name** – Name of the VPN customer for which the list of PEs is desired.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - PE: router struct
  - vpnState: state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2])) {
    printf "usage: RouteAnalyzer.api_vpn_customer_pe_list ip database
customer\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $customer = $ARGV[2];
my $filter = "any";
```

```
$filter = $ARGV[3] if ($#ARGV >= 3);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("30 Aug 2005 00:26:30 PDT");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_pe_list',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($customer),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print (STDERR join "\n", "---XMLRPC RESULT value1 ---", Dumper($value1) );
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'VOD',
```

```
  'report_time' => '20051115T19:14:20',
  'totalEntries' => '1',
  'result' => [
    {
      'PE' => {
        'type' => 'Originator',
        'ipaddr' => '192.168.180.180'
      },
      'vpnState' => {
        'inBaseline' => 'false',
        'down' => 'true'
      }
    }
  ]
}
```

api_vpn_customer_reachability

RPC Call: RouteAnalyzer.api_vpn_customer_reachability {password} {database name} {time} {filter}

This query returns reachability statistics for each VPN customer. Reachability is specified in terms of the percentage deviation from the baseline reachability. For example, this could be negative if some routes are down and fewer routes are available than those at baseline. This could be positive if new routes have been added that were not known at baseline.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - customer: string
  - definition: string
  - numPEs: int
  - numActiveRoutes: int
  - numBaselineRoutes: int
  - numDownRoutes: int
  - numNewRoutes: int
  - deviation: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_customer_reachability ip
database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
```

```
$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Feb 2005 15:50:22 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_reachability',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
---XMLRPC RESULT value1 ---

{
    'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '50',
  'network_name' => 'VOD',
```

```
  'report_time' => '20051115T19:19:49',
  'totalEntries' => '50',
  'result' => [
    {
      'numDownRoutes' => '1',
      'numActiveRoutes' => '0',
      'numNewRoutes' => '0',
      'numPEs' => '0',
      'customer' => 'Cust747',
      'deviation' => '100',
      'numBaselineRoutes' => '1',
      'definition' => 'RT:600:1'
    }
   ....
  ]
}
```

api_vpn_customer_reachability_by_peer

RPC Call: RouteAnalyzer.api_vpn_customer_reachability_by_peer {password} {database name} {time} {customer name} {filter}

This query returns reachability statistics at each PE for the specified VPN customer. Reachability is specified in terms of the percentage deviation from the baseline reachability. For example, this could be negative if some routes are down and fewer routes are available than those at baseline.
This could be positive if new routes have been added that were not known at baseline.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **customer name** – Name of the VPN customer for which reachability information is desired.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - PE: router struct
  - vpnState: state struct (with baseline)
  - numActiveRoutes: int
  - numBaselineRoutes: int
  - numDownRoutes: int
  - numNewRoutes: int
  - deviation: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2])) {
    printf "usage: RouteAnalyzer.api_vpn_customer_reachability_by_peer ip
database customer\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
```

```
my $customer = $ARGV[2];
my $filter = "any";

$filter = $ARGV[3] if ($#ARGV >= 3);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("30 Aug 2005 00:26:30 PDT");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_reachability_by_pe
er',
RPC::XML::RPC_STRING($password),
RPC::XML::RPC_STRING($database),
RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
RPC::XML::RPC_STRING($customer),
RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print (STDERR join "\n", "---XMLRPC RESULT value1 ---", Dumper($value1) );
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
```

```
  },
  'numReturnedEntries' => '25',
  'network_name' => 'VOD',
  'report_time' => '20051115T19:12:35',
  'totalEntries' => '25',
  'result' => [
    {
      'numDownRoutes' => '0',
      'numActiveRoutes' => '1',
      'numNewRoutes' => '0',
      'PE' => {
        'type' => 'Originator',
        'ipaddr' => '192.168.180.180'
      },
      'deviation' => '0',
      'numBaselineRoutes' => '1',
      'vpnState' => {
        'inBaseline' => 'false',
        'down' => 'true'
      }
    }
  ]
}
```

api_vpn_route_target_pe_participation

RPC Call: RouteAnalyzer.api_vpn_route_target_pe_participation {password} {database name} {time} {filter}

This query returns statistics of participating PEs for each route target in the specified network. This includes information about the route target, the deviation from baseline, and the number of PEs that are active, down, or newly added after baseline.

Input Parameters

- **password** – The password configured for queries.

- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - routeTarget: string
  - numActivePEs: int
  - numBaselinePEs: int
  - numDownPEs: int
  - numNewPEs: int
  - deviation: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_route_target_pe_participation ip
database\n";
    exit(0);
}
```

```
my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Feb 2005 15:50:22 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_pe_participati
on',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '50',
  'network_name' => 'pd353',
  'report_time' => '20051028T00:23:42',
  'totalEntries' => '50',
  'result' => [
    {
      'routeTarget' => 'RT:65522:600',
      'numActivePEs' => '3',
      'deviation' => '100',
      'numNewPEs' => '3',
      'numDownPEs' => '0',
      'numBaselinePEs' => '0'
    },
    {
      'routeTarget' => 'RT:65522:2300',
      'numActivePEs' => '1',
      'deviation' => '100',
      'numNewPEs' => '1',
      'numDownPEs' => '0',
      'numBaselinePEs' => '0'
    },
    {
      'routeTarget' => 'RT:65522:500',
      'numActivePEs' => '2',
      'deviation' => '100',
      'numNewPEs' => '2',
      'numDownPEs' => '0',
      'numBaselinePEs' => '0'
    },
    {
      'routeTarget' => 'RT:65522:1500',
      'numActivePEs' => '2',
      'deviation' => '100',
      'numNewPEs' => '2',
```

```
        'numDownPEs' => '0',
        'numBaselinePEs' => '0'
      },
      ....
    ]
}
```

api_vpn_route_target_pe_list

RPC Call: RouteAnalyzer.api_vpn_route_target_privacy_by_peer {password} {database name} {time} {route target} {filter}

This query returns the list of participating PE routers and their VPN state for the specified route target.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **route target** – A label specifying the route target of interest (for example, RT:600:1).
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time

- totalEntries: int
- result: array of the following:
  - PE: router struct
  - vpnState: state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2])) {
    printf "usage: RouteAnalyzer.api_vpn_route_target_pe_list ip database
route-target\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $route_target = $ARGV[2];
my $filter = "any";

$filter = $ARGV[3] if ($#ARGV >= 3);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Aug 2005 15:50:22 PDT");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_pe_list',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($route_target),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
```

```
print (STDERR join "\n", "---XMLRPC RESULT value1 ---", Dumper($value1) );
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '25',
  'network_name' => 'VOD',
  'report_time' => '20051108T19:51:50',
  'totalEntries' => '25',
  'result' => [
    {
      'PE' => {
        'type' => 'Originator',
        'ipaddr' => '192.168.180.180'
      },
      'vpnState' => {
        'inBaseline' => 'false',
        'down' => 'true'
      }
    }
  ]
}
```

api_vpn_route_target_reachability

RPC Call: RouteAnalyzer.api_vpn_route_target_reachability {password} {database name} {time} {filter}

This query returns reachability statistics for each route target in the specified network. This includes information about the deviation from baseline and the number of routes that are down, active, and newly added after the baseline.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- TotalEntries: int
- result: array of the following:
  - routeTarget: string
  - numPEs: int
  - numActiveRoutes: int
  - numBaselineRoutes: int
  - numDownRoutes: int
  - numNewRoutes: int
  - deviation: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_route_target_reachability ip
database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
my $t1 = str2time("28 Jul 2004 08:25:51 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_reachability',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
print Dumper($value1);
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '50',
  'network_name' => 'pd353',
  'report_time' => '20051027T23:25:19',
  'totalEntries' => '50',
  'result' => [
    {
      'routeTarget' => 'RT:65522:100',
      'numDownRoutes' => '0',
      'numActiveRoutes' => '0',
      'numPEs' => '0',
      'numNewRoutes' => '0',
      'deviation' => '100',
      'numBaselineRoutes' => '0'
    },
    {
      'routeTarget' => 'RT:65522:600',
      'numDownRoutes' => '0',
      'numActiveRoutes' => '0',
      'numPEs' => '0',
      'numNewRoutes' => '0',
      'deviation' => '100',
      'numBaselineRoutes' => '0'
    },
    {
      'routeTarget' => 'RT:65522:2400',
      'numDownRoutes' => '0',
      'numActiveRoutes' => '0',
      'numPEs' => '0',
      'numNewRoutes' => '0',
      'deviation' => '100',
      'numBaselineRoutes' => '0'
    },
    {
      'routeTarget' => 'RT:65522:700',
```

```
        'numDownRoutes' => '0',
        'numActiveRoutes' => '0',
        'numPEs' => '0',
        'numNewRoutes' => '0',
        'deviation' => '100',
        'numBaselineRoutes' => '0'
      }
      ....
  ]
}
```

api_vpn_route_target_reachability_by_peer

RPC Call: RouteAnalyzer.api_vpn_route_target_reachability_by_peer {password} {database name} {time} {route target} {filter}

This query returns reachability statistics at each PE for the specified route target. This includes information about the deviation from baseline and the number of routes that are down, active, and newly added after the baseline.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **route target** – A label specifying the route target of interest (for example, RT:600:1).
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: version struct

- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - PE: router struct
  - vpnState: state struct (with baseline)
  - numActiveRoutes: int
  - numBaselineRoutes: int
  - numDownRoutes: int
  - numNewRoutes: int
  - deviation: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2])) {
    printf "usage: RouteAnalyzer.api_vpn_route_target_reachability_by_peer
ip database route_target\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
my $route_target = $ARGV[2];

$filter = $ARGV[3] if ($#ARGV >= 3);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Aug 2005 16:16:45 PDT");


push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_reachability_b
y_peer',
RPC::XML::RPC_STRING($password),
RPC::XML::RPC_STRING($database),
RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
RPC::XML::RPC_STRING($route_target),
RPC::XML::RPC_STRING($filter) ));

foreach (@reqs) {
my $res = $client->send_request($_);
```

```
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print (STDERR join "\n", "---XMLRPC RESULT value1 ---", Dumper($value1) );
}
}
```

Sample Output

```
---XMLRPC RESULT value1 ---
{
  'vinfo' => {
    'software_version' => ''8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '20',
  'network_name' => 'VOD',
  'report_time' => '20051108T20:04:47',
  'totalEntries' => '20',
  'result' => [
    {
      'numDownRoutes' => '0',
      'numActiveRoutes' => '1',
      'numNewRoutes' => '1',
      'PE' => {
        'type' => 'Originator',
        'ipaddr' => '192.168.180.180'
      },
      'deviation' => '100',
      'numBaselineRoutes' => '0',
      'vpnState' => {
        'inBaseline' => 'false',
        'down' => 'true'
      }
    }
  ]
}
```

api_vpn_routes

RPC Call: RouteAnalyzer.api_vpn_routes {password} {database name} {time} {filter}

This query returns the list of VPN routes for the specified network.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: verstion struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: array of the following:
  - topology: topology struct
  - vpnPrefix:
    - —labelStack: string
    - —prefix: string
  - attributes: BGP attribute struct
  - router: router struct

— state: state struct (with baseline)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_vpn_routes ip database\n";
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("28 Feb 2005 15:50:22 PST");

push (@reqs, RPC::XML::request->new('RouteAnalyzer.api_vpn_routes',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING($filter) ));
foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
---XMLRPC RESULT value1 ---

  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '20',
  'network_name' => 'pd353',
  'report_time' => '20051027T21:40:07',
  'totalEntries' => '20',
  'result' => [
    {
      'topology' => {
        'fullName' => 'pd353.Left.BGP/AS65522/VPN',
        'protocol' => 'BGP'
      },
      'vpnPrefix' => {
        'labelStack' => '20543',
        'prefix' => '65522:700:192.168.230.230/24'
      },
      'attributes' => {
        'mpReachabilityNextHop' => '0:192.168.104.12',
        'extCommunities' => 'RT:65522:700 ',
        'origin' => 'INCOMPLETE',
        'localPref' => '100',
        'asPath' => '',
        'med' => '0'
      },
      'router' => {
        'type' => 'IBGP Peer',
        'ipaddr' => '192.168.200.200'
      },
      'state' => {
        'inBaseline' => 'false',
        'down' => 'false'
      }
    },
    ....

  ]
```

```
}
```

api_vpn_routes_handle

RPC Call: RouteAnalyzer.api_vpn_routes_handle {password} {database} {time} {filter}

This query returns a handle for the list of VPN routes for the specified network.

Input Parameters

- **password** – The password configured for queries.
- **database name** – One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.BGP/AS65522/VPN.
- **time** – A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **filter** – A filter expression to limit the results to the subset matching the filter parameters. See the "History Navigator" chapter in the *HP Route Analytics Management System User's Guide* for more information about filter expressions. Use the filter "any" to return the full results.

Structure of Output

- vinfo: verstion struct

- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- result: int

Example and Sample Output

See explanation of re-entrant queries in Using Re-Entrant Queries on page 39.

5 VPN Customer Report Queries

The queries in this chapter require a RAMS Traffic system with an MPLS VPN license. In addition, these queries are enabled only if the system is licensed for the VPN Customer Reports feature.

This chapter describes the calls, input parameters and results for RAMS Traffic XML RPC queries. These queries are used to generate VPN customer reports that the Service Provider can generate per Enterprise customer.

For details regarding how to configure these reports, see the "VPN Routing" chapter in the *HP Route Analytics Management System User's Guide*.

The following queries are includes in this chapter:

- api_traffic_vpn_customer_wan_connection_topn_srcs
- api_vpn_customer_default_reporting_wan_connections_get
- api_vpn_customer_default_reporting_wan_connections_set
- api_vpn_customer_wan_connection_get_config
- api_vpn_customer_wan_connection_set_config
- api_vpn_enabled_customer_list_get_config

api_traffic_vpn_customer

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer {password} {database name} {time} {report time range} {customer name} {wan connection name}

This query returns the aggregate traffic statistics for a VPN customer.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection name—** (This parameter is optional) A WAN connection name to filter the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- total entries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- customer report result: array of the following structures:
  - customer_name: string
  - ingress_avg (bps)
  - ingress_min (bps)
  - ingress_max (bps)
  - ingress_ninetyfifthpctile (bps)
  - egress_avg (bps)
  - egress_min (bps)
  - egress_max (bps)
  - egress_ninetyfifthpctile (bps)

Example

```
#!/usr/bin/perl
if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}
my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
$filter = $ARGV[2] if ($#ARGV >= 2);
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
```

```
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
my $t1 = str2time("20080922T12:30:00");
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );
foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
print Dumper($value1);
}
```

Sample Output

```
TrafficAnalyzer.api_traffic_vpn_customer:
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20080927T20:57:40',
  'totalEntries' => '1',
  'result' => {
    'report_result' => [
      {
        'avg' => '3441361',
        'min' => '1595297',
        'max' => '7664250',
        'ninetyfifthpctile' => '7664250',
        'customer_name' => 'COLA'
```

```
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
  }
```

api_traffic_vpn_customer_cos

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_cos {password} {database name} {time} {report time range} {customer name} {wan connection name}

This query returns breakdown of the aggregate traffic statistics by CoS group.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection name—** (This parameter is optional) A WAN connection name to filter the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- customer cos: array of the following structures:
  - customer_name: string
  - cos: string

- avg: double (bps)
- min: double (bps)
- max: double (bps)
- ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_cos',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );
```

```
foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20080927T20:55:24',
  'totalEntries' => '6',
  'result' => {
    'report_result' => [
      {
        'cos' => 'expZero',
        'avg' => '2202160',
        'min' => '416347',
        'max' => '6332036',
        'ninetyfifthpctile' => '6332036',
        'customer_name' => 'COLA'
      },
      {
        'cos' => 'Exp4',
        'avg' => '625616',
        'min' => '496134',
        'max' => '885887',
        'ninetyfifthpctile' => '885887',
        'customer_name' => 'COLA'
      },
      {
        'cos' => 'Exp2',
        'avg' => '191298',
        'min' => '163814',
        'max' => '265402',
        'ninetyfifthpctile' => '265402',
```

```
          'customer_name' => 'COLA'
        },
        {
          'cos' => 'Exp3',
          'avg' => '191006',
          'min' => '155474',
          'max' => '218288',
          'ninetyfifthpctile' => '218288',
          'customer_name' => 'COLA'
        },
        {
          'cos' => 'Exp1',
          'avg' => '117205',
          'min' => '31232',
          'max' => '176700',
          'ninetyfifthpctile' => '176700',
          'customer_name' => 'COLA'
        },
        {
          'cos' => 'Exp6',
          'avg' => '114073',
          'min' => '91737',
          'max' => '172351',
          'ninetyfifthpctile' => '172351',
          'customer_name' => 'COLA'
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
  }
```

api_traffic_vpn_customer_cos_history

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_cos_history {password} {database name} {start time} {end time} {customer name} {cos} {type of stats} {report time range)

This query returns the history for the type of statistic (minimum, maximum, average) for the VPN customer, the CoS group, and for a given time period.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **end time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **customer name**—The name of the VPN customer.
- **cos**—The Class of Service for the customer.
- **type of stats**—Displayed statistics: minimum (min), maximum (max), average (avg), or percentile (all case-insensitive).
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string

- end_time: ISO 8601 UTC time
- cos: string
- customer: string

- customer cos history: array of the following structures
  - time: ISO 8601 UTC time
  - type_of_data: int (bps)
- start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $startTime = str2time("20080710T16:00:00PST");
my $endTime = str2time("20080814T16:00:00PST");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_cos_hist
ory',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("COLA"),
                            RPC::XML::RPC_STRING("Exp1"),
```

```
                                RPC::XML::RPC_STRING("Average"),
                                RPC::XML::RPC_STRING("daily"))
                                );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '0',
  'network_name' => 'PDlab',
  'report_time' => '20080929T22:43:36',
  'totalEntries' => '0',
  'result' => {
    'history_vpn_customer_cos' => {
      'end_time' => '20080814T06:59:59',
      'cos' => 'Exp1',
      'customer' => 'COLA',
      'statistics' => [
        {
          'time' => '20080731T07:00:00',
          'avg' => '102896'
        },
        {
          'time' => '20080801T07:00:00',
          'avg' => '85747'
        },
        {
          'time' => '20080802T07:00:00',
          'avg' => '100535'
        },
        {
          'time' => '20080803T07:00:00',
          'avg' => '87326'
        },
        {
          'time' => '20080804T07:00:00',
          'avg' => '107551'
        },
        {
          'time' => '20080805T07:00:00',
          'avg' => '106075'
        },
        {
```

```
          'time' => '20080806T04:00:00',
          'avg' => '102188'
        },
        {
          'time' => '20080807T07:00:00',
          'avg' => '7624'
        },
        {
          'time' => '20080808T07:00:00',
          'avg' => '8626'
        },
        {
          'time' => '20080809T06:50:00',
          'avg' => '6596'

        },
        {
          'time' => '20080813T07:00:00',
          'avg' => '96249'
        }
      ],
      'start_time' => '20080711T00:00:00'
    }
  }
}
```

api_traffic_vpn_customer_history

RPC Call: TrafficAnalyzer_api_traffic_vpn_customer_history {password} {database name} {start time} {end time} {customer name} {type of stats}{report time range}

This query returns the history statistics for the VPN customer for the given time period.

Input Parameters

- **password**—The password configured for the queries.
- **database name**—One or more space-separated names in the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. This is the start of the interval for the historical time frame in question.

- **end time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.This is the end of the interval for the historical time frame in question.
- **customer name**—The name of the VPN customer.
- **type of stats**—Displayed statistics: minimum (min), maximum (max), average (avg), or percentile (all case-insensitive).
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- customer: string
- customer history: array of the following structures:
  - time: ISO 8601 UTC time
  - type of data: int (bps)
- start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $startTime = str2time("20080710T16:00:00PST");
my $endTime = str2time("20080814T16:00:00PST");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("COLA"),
                            RPC::XML::RPC_STRING("Average"),
                            RPC::XML::RPC_STRING("daily"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
```

```
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '11',
  'network_name' => 'PDlab',
  'report_time' => '20080929T22:45:05',
  'totalEntries' => '11',
  'result' => {
    'history_vpn_customer' => {
      'end_time' => '20080814T06:59:59',
      'customer' => 'COLA',
      'statistics' => [
        {
          'time' => '20080731T07:00:00',
          'avg' => '710624'
        },
        {
          'time' => '20080801T07:00:00',
          'avg' => '801396'
        },
        {
          'time' => '20080802T07:00:00',
          'avg' => '963687'
        },
        {
          'time' => '20080803T07:00:00',
          'avg' => '635650'
        },
        {
          'time' => '20080804T07:00:00',
          'avg' => '748942'
        },
{
          'time' => '20080805T07:00:00',
          'avg' => '718682'
        },
        {
          'time' => '20080806T07:00:00',
          'avg' => '765568'
        },
        {
          'time' => '20080807T07:00:00',
```

```
            'avg' => '1071895'
          },
          {
            'time' => '20080808T07:00:00',
            'avg' => '986013'
          },
          {
            'time' => '20080809T07:00:00',
            'avg' => '1048586'
          },
          {
            'time' => '20080813T07:00:00',
            'avg' => '942569'
          }
        ],
        'start_time' => '20080711T00:00:00'
      }
    }
}
```

api_traffic_vpn_customer_wan_connection

RPC Call: TrafficAnalyzer_vpn_customer_wan_connection {password} {database name} {time} {report time range} {customer name} {wan connection filter}

This query returns ingress and egress traffic statistics for traffic going to and from all the WAN connections belonging to a particular VPN customer, and for a particular period of time.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection filter—**(This parameter is optional) A WAN connection filter to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- wan connections: array of the following structures:
  - customer_name: string
  - wan_name: string
  - ingress_avg (bps)
  - ingress_min (bps)
  - ingress_max (bps)
  - ingress_ninetyfifthpctile (bps)
  - egress_min (bps)
  - egress_max (bps)
  - egress_ninetyfifthpctile (bps)

Example

```
#!/usr/bin/perl
```

```
if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output


```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:05:25',
  'totalEntries' => '5',
  'result' => {
    'report_result' => [
      {
        'egress_min' => '-1',
        'wan_connection_name' => 'Check21',
        'ingress_ninetyfifthpctile' => '5878141',
        'ingress_min' => '0',
        'ingress_max' => '5878141',
        'egress_avg' => '-1',
        'egress_ninetyfifthpctile' => '-1',
        'ingress_avg' => '1769069',
        'customer_name' => 'COLA',
        'egress_max' => '-1'
      },
      {
        'egress_min' => '-1',
        'wan_connection_name' => 'West Coast',
        'ingress_ninetyfifthpctile' => '1390252',
        'ingress_min' => '1059136',
        'ingress_max' => '1390252',
        'egress_avg' => '-1',
        'egress_ninetyfifthpctile' => '-1',
        'ingress_avg' => '1160431',
        'customer_name' => 'COLA',
        'egress_max' => '-1'
      },
      {
        'egress_min' => '-1',
        'wan_connection_name' => 'SecondWestCoast',
        'ingress_ninetyfifthpctile' => '708178',
        'ingress_min' => '393791',
```

```
          'ingress_max' => '708178',
          'egress_avg' => '-1',
          'egress_ninetyfifthpctile' => '-1',
          'ingress_avg' => '511860',
          'customer_name' => 'COLA',
          'egress_max' => '-1'
        },
        {
          'egress_min' => '1539286',
          'wan_connection_name' => 'East Coast',
          'ingress_ninetyfifthpctile' => '-1',
          'ingress_min' => '-1',
          'ingress_max' => '-1',
          'egress_avg' => '1665875',
          'egress_ninetyfifthpctile' => '2098430',
          'ingress_avg' => '-1',
          'customer_name' => 'COLA',
          'egress_max' => '2098430'
        },
        {
          'egress_min' => '0',
          'wan_connection_name' => 'UNKNOWN',
          'ingress_ninetyfifthpctile' => '-1',
          'ingress_min' => '-1',
          'ingress_max' => '-1',
          'egress_avg' => '1775485',
          'egress_ninetyfifthpctile' => '5878141',
          'ingress_avg' => '-1',
          'customer_name' => 'COLA',
          'egress_max' => '5878141'
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
  }
```

api_traffic_vpn_customer_wan_connection_cos

RPC Call: TrafficAnalyzer.api_vpn_customer_wan_connection_cos {password} {database name} {time} {report time range} {customer name}{wan connection filter}

This query returns the traffic breakdown by CoS group for each WAN connection within a given VPN customer.

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection filter**—(This parameter is optional) A WAN connection name to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- wan connections: array of the following structures:
  - customer_name: string
  - wan_connection_name: string

- cos: string
- ingress_avg (bps)
- ingress_min (bps)
- ingress_max (bps)
- ingress_ninetyfifthpctile (bps)
- egress_avg (bps)
- egress_min (bps)
- egress_max (bps)
- egress_ninetyfifthpctile (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_cos',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:09:23',
  'totalEntries' => '5',
  'result' => {
    'report_result' => [
      {
        'wan_connection_name' => 'UNKNOWN',
        'ingress_ninetyfifthpctile' => '-1',
        'ingress_max' => '-1',
        'egress_avg' => '1774758',
        'egress_min' => '0',
        'cos' => 'expZero',
        'ingress_min' => '-1',
        'egress_ninetyfifthpctile' => '5878141',
        'ingress_avg' => '-1',
        'customer_name' => 'COLA',
```

```
  'egress_max' => '5878141'
},
{
  'wan_connection_name' => 'East Coast',
  'ingress_ninetyfifthpctile' => '-1',
  'ingress_max' => '-1',
  'egress_avg' => '427401',
  'egress_min' => '372626',
  'cos' => 'expZero',
  'ingress_min' => '-1',
  'egress_ninetyfifthpctile' => '453894',
  'ingress_avg' => '-1',
  'customer_name' => 'COLA',
  'egress_max' => '453894'
},
{
  'wan_connection_name' => 'UNKNOWN',
  'ingress_ninetyfifthpctile' => '-1',
  'ingress_max' => '-1',
  'egress_avg' => '727',
  'egress_min' => '0',
  'cos' => 'Exp6',
  'ingress_min' => '-1',
  'egress_ninetyfifthpctile' => '8726',
  'ingress_avg' => '-1',
  'customer_name' => 'COLA',
  'egress_max' => '8726'
},
{
  'wan_connection_name' => 'East Coast',
  'ingress_ninetyfifthpctile' => '-1',
  'ingress_max' => '-1',
  'egress_avg' => '117205',
  'egress_min' => '31232',
  'cos' => 'Exp1',
  'ingress_min' => '-1',
  'egress_ninetyfifthpctile' => '176700',
  'ingress_avg' => '-1',
  'customer_name' => 'COLA',
  'egress_max' => '176700'
},
{
```

```
        'wan_connection_name' => 'East Coast',
        'ingress_ninetyfifthpctile' => '-1',
        'ingress_max' => '-1',
        'egress_avg' => '191298',
        'egress_min' => '163814',
        'cos' => 'Exp2',
        'ingress_min' => '-1',
        'egress_ninetyfifthpctile' => '265402',
        'ingress_avg' => '-1',
        'customer_name' => 'COLA',
        'egress_max' => '265402'
      }
    ],
    'report_start_time' => '20080922T18:00:00',
    'report_end_time' => '20080922T18:59:59'
  }
}
```

api_traffic_vpn_customer_wan_connection_cos_history

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_cos_history {password} {database name} {start time} {end time} {customer name} {wan connection name} {cos}{type of stats}{type of data} {report time range}

This query returns the history of ingress or egress statistics (minimum, maximum, average) for the customer, WAN connection, and CoS group for the given time period.

Input Parameters

- **password**—The password configured for queries.
- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. This is the start of the interval for the historical time frame in question.
- **end time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.This is the end of the interval for the historical time frame in question.
- **customer name**—The name of the VPN customer.
- **wan connection name**—The name of the WAN connection belonging to the customer whose history is being viewed.
- **cos**—The Class of Service for the customer.
- **type of stats**—Displayed statistics: minimum (min), maximum (max), average (avg), or percentile (all case-insensitive).
- **type of data**—Specifies whether ingress or egress data is to be displayed.
- **report time range**—The statistical time range the report will retrieve. The time range can be any, hourly, daily, or monthly.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string

- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- cos: string
- wan_connection name: string
- customer: string
- customer wan connection cos history: array of the following structures:
  - time: ISO 8601 UTC time
  - type of data: int (bps)
- start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $startTime = str2time("20080710T16:00:00PST");
my $endTime = str2time("20080814T16:00:00PST");
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_cos_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("COLA"),
                            RPC::XML::RPC_STRING("Check21"),
                            RPC::XML::RPC_STRING("Exp1"),
                            RPC::XML::RPC_STRING("Average"),
                            RPC::XML::RPC_STRING("ingress"),
                            RPC::XML::RPC_STRING("daily"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20080929T22:40:44',
  'totalEntries' => '6',
  'result' => {
    'history_vpn_customer_wan_connection_cos' => {
      'end_time' => '20080814T23:59:59',
      'cos' => 'Exp5',
      'wan_connection' => 'Check21',
      'customer' => 'COLA',
```

```
        'statistics' => [
          {
            'time' => '20080812T16:00:00',
            'avg' => '2870'
          },
          {
            'time' => '20080812T17:00:00',
            'avg' => '35142'
          },
          {
            'time' => '20080812T18:00:00',
            'avg' => '30786'
          },
          {
            'time' => '20080812T22:00:00',
            'avg' => '2276'
          },
          {
            'time' => '20080812T23:00:00',
            'avg' => '70323'
          },
          {
            'time' => '20080813T00:00:00',
            'avg' => '68672'
          }
        ],
        'start_time' => '20080711T00:00:00'
      }
    }
}
```

api_traffic_vpn_customer_wan_connection_history

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_history {password} {database name} {start time} {end time} {customer name} {wan connection name} {type of stats) {report time range}

This query returns the history of ingress or ingress statistics (minimum, maximum, average) for the VPN customer and WAN connection for the given time period.

Input Parameters

- **password**—The password configured for queries.
- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. This is the start of the interval for the historical time frame in question.
- **end time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.This is the end of the interval for the historical time frame in question.
- **customer name**—The name of the VPN customer.
- **wan connection name**—The name of the WAN connection belonging to the customer.
- **type of stats**—Displayed statistics: minimum (min), maximum (max), average (avg), or percentile (all case-insensitive).
- **report time range**—The statistical time range the report will retrieve. The time range can be any, hourly, daily, or monthly.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- wan_connection_name: string
- customer: string
- customer wan connection history: array of the following structures:

- time: ISO 8601 UTC time
- type of data: int (bps)

- start_ time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $startTime = str2time("20080710T16:00:00PST");
my $endTime = str2time("20080814T16:00:00PST");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("COLA"),
                            RPC::XML::RPC_STRING("Check21"),
                            RPC::XML::RPC_STRING("Average"),
```

```
                                RPC::XML::RPC_STRING("ingress"),
                                RPC::XML::RPC_STRING("daily"))
                                );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
</params>
</methodCall>". $EOL;
while (<$remote>) { print; }
close $remote;
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20080929T22:41:48',
  'totalEntries' => '6',
  'result' => {
    'history_vpn_customer_wan_connection' => {
      'end_time' => '20080814T23:59:59',
      'wan_connection' => 'Check21',
      'customer' => 'COLA',
      'statistics' => [
        {
          'time' => '20080812T16:00:00',
          'avg' => '64556'
        },
        {
          'time' => '20080812T17:00:00',
          'avg' => '751503'
        },
        {
```

```
            'time' => '20080812T18:00:00',
            'avg' => '681259'
          },
          {
            'time' => '20080812T22:00:00',
            'avg' => '33042'
          },
          {
            'time' => '20080812T23:00:00',
            'avg' => '885098'
          },
          {
            'time' => '20080813T00:00:00',
            'avg' => '1012338'
          }
        ],
        'start_time' => '20080711T00:00:00'
      }
    }
}
```

api_traffic_vpn_customer_wan_connection_to_wan_connection

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_to_wan_connection {password} {database name} {time} {report time range} {customer name} {source wan connection filter} {destination wan connection filter}

This query returns statistics for VPN traffic between the 100 selected WAN connections. These reported statistics are for the traffic from the source WAN connection to the destination WAN connection.

Input Parameters

- **password**—The password configured for queries.

- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **source wan connection filter**—(this is an optional parameter) This filters traffic from the source WAN connection.
- **destination wan connection filter**—(this is an optional parameter) This filters traffic from the destination WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- wan_connection_to_wan_connection: array of the following structures:

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}
my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
$filter = $ARGV[2] if ($#ARGV >= 2);
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
my $t1 = str2time("20080922T12:30:00");
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_to_wan_connection',
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:13:44',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'source_wan_connection_name' => 'East Coast',
        'destination_wan_connection_name' => 'West Coast',
```

```
          'avg' => '1154014',
          'min' => '1042977',
          'max' => '1390252',
          'customer_name' => 'COLA'
        },
        {
          'source_wan_connection_name' => 'East Coast',
          'destination_wan_connection_name' => 'SecondWestCoast',
          'avg' => '511860',
          'min' => '393791',
          'max' => '708178',
          'customer_name' => 'COLA'
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
  }
```

api_traffic_vpn_customer_wan_connection_to_wan_connection_history

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_to_wan_connection {password} {database name}{start time} {end time} {customer name} {source_wan_connection} {destination_wan_connection} {type of stats} {report time range}

This query returns the history of ingress or egress statistics (minimum, maximum, average) in bps for the VPN customer source and destination WAN connection provided for a given time period.

Input Parameters

- **password**—The password configured for queries.
- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain, such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **start time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. This is the start of the interval for the historical time frame in question.
- **end time**—Time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.This is the end of the interval for the historical time frame in question.
- **customer name**—The name of the VPN customer.
- **src wan connection name**—The name of the source WAN connection belonging to the VPN customer the history statistics are being returned for.
- **dst wan connection name**—The name of the destination WAN connection belonging to the VPN customer the history statistics are being returned for.
- **type of stats**—Displayed statistics: minimum (min), maximum (max), average (avg), or percentile (all case-insensitive).
- **report time range**—The statistical time range the report will retrieve. The time range can be any, hourly, daily, or monthly.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- src_wan_connection Name: string
- end_time: ISO 8601 UTC time
- customer: string
- wan connection to wan connection history: array of the following structures:
  - time: ISO 8601 UTC time
  - type of data: int (bps)
- start_time: ISO 8601 UTC time
- dst_wan_connection Name: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $startTime = str2time("20080710T16:00:00PST");
my $endTime = str2time("20080814T16:00:00PST");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_to_wan_connection_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("COLA"),
                            RPC::XML::RPC_STRING("Check21"),
                            RPC::XML::RPC_STRING("10.120.1.7_East Coast"),
                            RPC::XML::RPC_STRING("Average"),
                            RPC::XML::RPC_STRING("daily"))
                            );

foreach (@reqs) {
```

```
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => ' 8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20080929T22:38:21',
  'totalEntries' => '6',
  'result' => {
    'history_vpn_customer_wan_connection_to_wan_connection' => {
      'src_wan_connection' => 'Check21',
      'end_time' => '20080814T23:59:59',
      'customer' => 'COLA',
      'statistics' => [
        {
          'Time' => '20080812T16:00:00',
          'avg' => '1282'
        },
        {
          'Time' => '20080812T17:00:00',
          'avg' => '8842'
        },
        {
          'Time' => '20080812T18:00:00',
          'avg' => '476'
        },
        {
          'Time' => '20080812T22:00:00',
          'avg' => '424'
        },
        {
          'Time' => '20080812T23:00:00',
```

```
            'avg' => '7429'
          },
          {
            'Time' => '20080813T00:00:00',
            'avg' => '2900'
          }
        ],
        'start_time' => '20080711T00:00:00',
        'dst_wan_connection' => '10.120.1.7_East Coast'
      }
    }
}
```

api_traffic_vpn_customer_wan_connection_topn_convers

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_topn_convers {password} {database name} {time) {report time range} {customer name) {wan connection name}

This query returns the average traffic statistics for the top 100 conversations between the source and destination addresses for a particular WAN connection.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.

- **wan connection name—**(This parameter is optional) A WAN connection name to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- topn conversations: array of the following structures:
  - customer_name: string
  - wan_connection_name: String
  - source_ip_address: IP address
  - destination_ip_address: IP address
  - Avg: integer (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
```

```
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_topn_convers',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:17:30',
  'totalEntries' => '5',
  'result' => {
    'report_result' => [
      {
```

```
          'source_address' => '172.16.34.1',
          'wan_connection_name' => 'East Coast',
          'destination_address' => '10.70.200.1',
          'avg' => '142',
          'customer_name' => 'COLA'
        },
        {
          'source_address' => '172.16.1.1',
          'wan_connection_name' => 'East Coast',
          'destination_address' => '10.70.102.1',
          'avg' => '33',
          'customer_name' => 'COLA'
        },
        {
          'source_address' => '172.16.3.1',
          'wan_connection_name' => 'East Coast',
          'destination_address' => '10.70.104.1',
          'avg' => '16',
          'customer_name' => 'COLA'
        },
        {
          'source_address' => '172.16.2.1',
          'wan_connection_name' => 'East Coast',
          'destination_address' => '10.70.103.1',
          'avg' => '16',
          'customer_name' => 'COLA'
        },
        {
          'source_address' => '172.16.4.1',
          'wan_connection_name' => 'East Coast',
          'destination_address' => '10.70.105.1',
          'avg' => '13',
          'customer_name' => 'COLA'
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
  }
```

api_traffic_vpn_customer_wan_connection_topn_dsts

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_topn_dsts {password} {database name} {time) {report time range} {customer name) {wan connection name}

This query returns the statistics for the top 100 destinations for a particular WAN connection.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection name—**(This parameter is optional) A WAN connection name to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- topn destinations: array of the following structures:
  - customer_name: string
  - wan_connection_name: string

— ip_address: string

— Avg: integer (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_topn_dsts',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );
```

```
foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '98',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:20:48',
  'totalEntries' => '98',
  'result' => {
    'report_result' => [
      {
        'wan_connection_name' => 'SecondWestCoast',
        'avg' => '142',
        'address' => '10.70.200.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'West Coast',
        'avg' => '35',
        'address' => '10.70.102.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'Check21',
        'avg' => '21',
        'address' => '172.17.4.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'Check21',
```

```
          'avg' => '21',
          'address' => '172.17.3.1',
          'customer_name' => 'COLA'
        },
        {
          'wan_connection_name' => 'Check21',
          'avg' => '-1',
          'address' => '172.17.39.1',
          'customer_name' => 'COLA'
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
}
```

api_traffic_vpn_customer_wan_connection_topn_port_protocol

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan_connection_topn_port_protocol {password} {database name} {time) {report time range} {customer name) {wan connection name}

This query returns traffic statistics for the top 100 protocol pairs for a particular WAN connection.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time—**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection name—**(This parameter is optional) A WAN connection name to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following structures:
  - customer_name: string
  - wan_connection_name: string
  - ip_address: string
  - avg: integer (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";
```

```
$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_topn_port_protocol',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
```

```
  'report_time' => '20080927T21:27:34',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'protocol' => '6',
        'wan_connection_name' => 'East Coast',
        'avg' => '377',
        'customer_name' => 'COLA',
        'port' => '23'
      },
      {
        'protocol' => '6',
        'wan_connection_name' => 'East Coast',
        'avg' => '85',
        'customer_name' => 'COLA',
        'port' => '49306'
      }
    ],
    'report_start_time' => '20080922T18:00:00',
    'report_end_time' => '20080922T18:59:59'
  }
}
```

api_traffic_vpn_customer_wan_connection_topn_srcs

RPC Call: TrafficAnalyzer.api_traffic_vpn_customer_wan connection_top_sources {password} {database name} {time) {report time range} {customer name) {wan connection name}

This query returns the traffic statistics for the top 100 sources for a particular WAN connection.

Input Parameters

- **password**—The password configured for queries.

- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistics are calculated. The time range can by hourly, daily, weekly, or monthly.
- **customer name**—The name of the VPN customer.
- **wan connection name**—(This parameter is optional) A WAN connection name to limit the results to output traffic statistics for a specified WAN connection.

Structure of Output

- vinfo: version struct
- network_name: String
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following structures:
  - customer_name: string
  - wan_connection_name: string
  - ip_address: string
  - avg: integer (bps)

Example

```
#!/usr/bin/perl
```

```
if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_customer_wan_conn
ection_topn_srcs',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
    RPC::XML::RPC_STRING("hourly"),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20080927T21:25:10',
  'totalEntries' => '5',
  'result' => {
    'report_result' => [
      {
        'wan_connection_name' => 'East Coast',
        'avg' => '142',
        'address' => '172.16.34.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'East Coast',
        'avg' => '37',
        'address' => '172.16.1.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'East Coast',
        'avg' => '31',
        'address' => '172.16.33.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'East Coast',
        'avg' => '20',
        'address' => '172.16.3.1',
        'customer_name' => 'COLA'
      },
      {
        'wan_connection_name' => 'East Coast',
        'avg' => '0',
        'address' => '172.16.32.1',
        'customer_name' => 'COLA'
```

```
        }
      ],
      'report_start_time' => '20080922T18:00:00',
      'report_end_time' => '20080922T18:59:59'
    }
}
```

api_vpn_customer_default_reporting_wan_connections_get

RPC Call: TrafficAnalyzer.api_vpn_customer_default_reporting_wan_connections_get {password} {database name} {customer}

This query returns a list of the WAN connections that are configured for calculating the Top N Reports (which can be accessed through the Top N APIs) and the WAN connection to WAN connection reports.

Input Parameters

- **password**—The password configured for queries.
- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain such as CorpNet, which includes the sub-tree below it. Or, it can be a complete database name, such as CorpNet.EIGRP/AS100.

- **customer**—The name of the VPN customer.

Structure of Output

- vinfo: version struct
- wan_connection_names: array of strings

  Each string in the array is the name of a specific WAN connection.

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_vpn_customer_default_reportin
g_wan_connections_get',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::RPC_STRING("COLA"))
    );
```

```
foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'wan_connection_names' => [
    'Check21',
    'SecondWestCoast',
    'West Coast',
    'East Coast',
    'SecondEastCoast'
  ]
}
```

api_vpn_customer_default_reporting_wan_connections_set

RPC Call: TrafficAnalyzer.api_vpn_customer_default_reporting_wan_connections_set {password} {database name} {customer} {wan connection names}

This query is used for setting a list of the WAN connections that will be used for calculating the Top N Reports (which can be accessed through the Top N APIs) and the WAN connection to WAN connection reports.

Input Parameters

- **password**—The password used for queries.

- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain such as CorpNet, which includes the sub-tree below it. Or, it can be a complete database name, such as CorpNet.EIGRP/AS100..
- **customer**—The name of the VPN customer.
- **wan connection names**—A structure containing the member "wan_connection_names," which has as its value an array of strings where each string is the name of the specific WAN connection.

Structure of Output

- vinfo: version struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
my $wan_connections_str = RPC::XML::struct->new (
'wan_connection_names' => RPC::XML::array->new(
             RPC::XML::string->new('SecondWestCoast'),
             RPC::XML::string->new('SecondEastCoast'),
             RPC::XML::string->new('West Coast'),
```

```
                RPC::XML::string->new('East Coast')
    )
    );

    push (@reqs,
    RPC::XML::request->new('TrafficAnalyzer.api_vpn_customer_default_reportin
    g_wan_connections_set',
        RPC::XML::RPC_STRING($password),
        RPC::XML::RPC_STRING($database),
        RPC::XML::RPC_STRING("COLA"),
        $wan_connections_str)
        );

    foreach (@reqs) {
    my $res = $client->send_request($_);
    if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
    my $value1 = $res->value;

    print Dumper($value1);
    }
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  }
}
```

api_vpn_customer_wan_connection_get_config

RPC Call: TrafficAnalyzer.api_vpn_customer_wan_connection_get_config {password} {database name} {customer name)

This query retrieves the configuration of the WAN connections for a specific customer.

Input Parameters

- **password**—The password configured for queries.
- **database name**—One or more space-separated names from the database hierarchy. Each name may be an administrative domain such as CorpNet, which includes the sub-tree below it. Or, it can be a complete database name, such as CorpNet.EIGRP/AS100.
- **customer name**—The name of the VPN customer.

Structure of Output

- vinfo: version struct
- wan_connections :
  - read_only_default_wan_connections: array of wan connection configuration
  - user_configured_wan_connections: array of wan connection configuration
- customer_pe_prefixes: array of PE prefix structures.

The output XML contains a structure with two members: "wan_connections" and "customer_pe_prefixes." The "wan_connections" member has as its value a structure with two members.

The first member has the name "read_only_default_wan_connections," which holds an array of WAN connection configurations. The WAN connections listed in this members' value are those that are internal to the system, and are provided here as a reference to build further WAN connection configurations. By default each PE, which is not part of any user-configured WAN connection configurations, is included here.

The second member has the name "user_configured_wan_connections," and has as its value an array of WAN connection configurations. This is similar to the member "read_only_default_wan_connections," except that the WAN connection configurations listed here are those that have been configured by a user. The "customer_pe_prefixes" member has as its value an array of PE Prefix structures. This member is provided as reference to allow you to build further WAN connection configurations.

A WAN connection configuration consists of a structure which has three members:

- wan_connection_name: string
- pe_list: array of strings (IP addresses of PEs which are part of this WAN connection)
- prefix_list: array of strings (prefixes which are part of this WAN connection)

A PE Prefix structure has the following two members:

- pe: string (IP address of the PE)
- prefix_list: array of strings (prefixes associated with the PE)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_vpn_customer_wan_connection_g
et_config',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::RPC_STRING("COLA"))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;
```

```
print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'customer_pe_prefixes' => [
    {
      'pe' => '10.120.1.7',
      'prefix_list' => [
        '10.130.1.33/32',
        '10.71.7.0/24',
        '10.71.8.0/24',
        '10.71.6.0/24',
        '10.130.1.27/32',
        '10.130.1.30/32',
        '1.1.1.0/24',
        '10.201.2.0/24',
        '10.91.1.0/24'
      ]
    },
    {
      'pe' => '10.120.1.9',
      'prefix_list' => [
        '10.111.2.1/32',
        '10.111.3.1/32',
        '10.120.25.1/32',
        '10.120.29.1/32',
        '10.120.29.2/32',
        '10.120.29.3/32'
      ]
    },
    {
      'pe' => '10.120.1.6',
      'prefix_list' => [
        '10.72.0.0/16',
        '10.132.1.0/24',
        '11.90.1.0/24',
        '11.90.2.0/24',
        '11.91.1.0/24',
        '11.92.1.0/24',
```

```
          '11.93.2.0/24',
          '11.93.3.0/24',
          '11.94.1.0/24',
          '10.70.1.0/30'
        ]
      },
      {
        'pe' => '10.120.1.8',
        'prefix_list' => [
          '10.70.103.1/32',
          '10.74.100.1/32',
          '10.74.100.2/32',
          '10.74.100.3/32',
          '10.74.100.4/32',
          '10.74.100.5/32',
          '10.74.100.6/32',
          '10.74.100.7/32',
          '10.74.100.8/32',
          '10.74.100.9/32'}
]
      }
    ],
    'wan_connections' => {
      'read_only_default_wan_connections' => [
        {
          'wan_connection_name' => 'SecondEastCoast',
          'pe_list' => [
            '10.120.1.9'
          ],
          'prefix_list' => []
        },
        {
          'wan_connection_name' => 'SecondWestCoast',
          'pe_list' => [
            '10.120.1.8'
          ],
          'prefix_list' => []
        },
        {
          'wan_connection_name' => 'East Coast',
          'pe_list' => [
            '10.120.1.7'
```

```
          ],
          'prefix_list' => []
        },
        {
          'wan_connection_name' => 'West Coast',
          'pe_list' => [
            '10.120.1.6'
          ],
          'prefix_list' => []
        }
      ],
      'user_configured_wan_connections' => [
        {
          'wan_connection_name' => 'Check21',
          'pe_list' => [
            '10.120.1.14'
          ],
          'prefix_list' => [
  '10.70.103.1/32'
]
        }
      ]
    }
}
```

api_vpn_customer_wan_connection_set_config

RPC Call: TrafficAnalyzer.api_vpn_customer_wan_connection_set_config {password} {database name} {customer name} {wan connection configuration}

This query is used to set the configuration for the WAN connections. This query will overwrite any previous configurations that may be present in the system.

Input Parameters

- **password**—The password configured for queries.

- **database name**—.One or more space-separated names from the database hierarchy. Each name may be an administrative domain such as CorpNet, which includes the sub-tree below it. Or, it can be a complete database name, such as CorpNet.EIGRP/AS100
- **customer name**—The name of the VPN customer.
- **wan connection configuration**—A nested structure in the same format as the output of the TrafficAnalyzer.api_vpn_customer_wan_connection_get_config, but occasionally omitting the members' "read_only_default_wan_connections" and "customer_pe_prefixes."

Structure of Output

- vinfo: version struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $wan_connection_str = RPC::XML::struct->new (
'wan_connections' => RPC::XML::struct->new(
'user_configured_wan_connections' => RPC::XML::array->new(
RPC::XML::struct->new(
```

```
'wan_connection_name' => RPC::XML::string->new('Check21'),
'pe_list' => RPC::XML::array->new(RPC::XML::string->new('10.120.1.14')),
'prefix_list' =>
RPC::XML::array->new(RPC::XML::string->new('10.70.103.1/32'))
)
)
)

);
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_vpn_customer_wan_connection_s
et_config',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database),
    RPC::XML::RPC_STRING("COLA"),
    $wan_connection_str)
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  }
}
```

api_vpn_enabled_customer_list_get_config

RPC Call: TrafficAnalyzer.api_vpn_enabled_customer_list_get_config {password} {database name}

This query returns an array of customer names that are enabled for reporting in the VPN Customer Configuration dialog.

Input Parameters

password—The password configured for queries.

database name—A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

Structure of Output

- vinfo: version struct
- result: array of strings

  Each string in the array is the name of a specific customer.

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_vpn_enabled_customer_list_get
_config',
    RPC::XML::RPC_STRING($password),
    RPC::XML::RPC_STRING($database))
    );

foreach (@reqs) {
my $res = $client->send_request($_);
if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
my $value1 = $res->value;

print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'result' => [
    'Bloomberg',
    'BostonCommunications',
    'Broadcom',
    'COLA',
    'Charter Communications',
    'ChinaTelecom',
    'Circuit city',
    'Cisco',
    'Citigroup',
    'Cognizant',
  ]
}
```

6 Traffic Report Queries

The queries in this chapter require a RAMS Traffic system.

This chapter describes the calls, input parameters and results for the following RAMS Traffic XML RPC queries:

- api_traffic_cos
- api_traffic_cos_history
- api_traffic_cos_ipv4
- api_traffic_cos_ipv4_history
- api_traffic_cos_vpn
- api_traffic_cos_vpn_history
- api_traffic_customers_cos_vpn
- api_traffic_destination_as_ipv4
- api_traffic_destination_as_ipv4_history
- api_traffic_egress_pe_vpn
- api_traffic_egress_router_ipv4
- api_traffic_egress_router_ipv4_history
- api_traffic_exporters
- api_traffic_exporters_history
- api_traffic_exporters_idx
- api_traffic_exporters_idx_history
- api_traffic_exporters_idx_ipv4
- api_traffic_exporters_idx_ipv4_history

- api_traffic_exporters_idx_vpn
- api_traffic_exporters_idx_vpn_history
- api_traffic_exporters_ipv4
- api_traffic_exporters_ipv4_history
- api_traffic_exporters_vpn
- api_traffic_exporters_vpn_history
- api_traffic_ingress_pe_vpn
- api_traffic_ingress_pe_vpn_history
- api_traffic_links
- api_traffic_links_cos
- api_traffic_links_cos_history
- api_traffic_links_cos_ipv4
- api_traffic_links_cos_ipv4_history
- api_traffic_links_cos_vpn
- api_traffic_links_cos_vpn_history
- api_traffic_links_history
- api_traffic_links_ipv4
- api_traffic_links_ipv4_history
- api_traffic_links_vpn
- api_traffic_links_vpn_history
- api_traffic_list_flows
- api_traffic_neighbor_as_ipv4
- api_traffic_neighbor_as_ipv4_history
- api_traffic_source_as_ipv4
- api_traffic_source_as_ipv4_history
- api_traffic_src_nbr_matrix
- api_traffic_src_dst_matrix
- api_traffic_traffic_groups_ipv4

- api_traffic_traffic_groups_ipv4_history
- api_traffic_transit_as_ipv4_history

api_traffic_cos

RPC Call: TrafficAnalyzer.api_traffic_cos {password} {database name} {time} {report time range}

This query returns the list of traffic statistics for aggregate Class of Service (CoS) for a particular time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- Report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- Array of CoS statistics structure containing the following:

- cos: string
- avg: double (bps)
- max: double (bps)
- ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs, RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                        RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;
```

```
        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:08:36',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'cos_group' => 'ANY',
        'avg' => '40272339',
        'min' => '37242156',
        'max' => '43185769',
        'ninetyfifthpctile' => '43185769'
      },
      {
        'cos_group' => 'Exp1',
        'avg' => '28763396',
        'min' => '25542590',
        'max' => '30577464',
        'ninetyfifthpctile' => '30577464'
      }
    ],
    'report_start_time' => '20081020T18:00:00',
    'report_end_time' => '20081020T18:59:59'
  }
}
```

api_traffic_cos_history

RPC Call: TrafficAnalyzer.api_traffic_cos_history {password} {database name) {start time} {end time} {cos} {statistics} {report time range}

This query returns the list of aggregate CoS (Class of Service) history data. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **cos—**The Class of Service (CoS) of the traffic.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- cos: string
- end_time: ISO 8601 UTC time

- array of following aggregate CoS history statistic structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $COS = $ARGV[5];
my $StatsType = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos_history',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($COS),
                            RPC::XML::RPC_STRING($StatsType),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:08:43',
  'totalEntries' => '1',
  'result' => {
    'history_cos' => {
      'end_time' => '20081015T22:00:00',
      'cos' => 'Exp1',
      'statistics' => [
        {
          'average' => '27764873',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_cos_ipv4

RPC Call: TrafficAnalyzer.api_traffic_cos_ipv4 {password} {database name} {time} {report time range}

This query returns the traffic statistics for IPv4 Class of Service (CoS) for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following CoS statistics structure :
  - cos: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
```

```
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:04:11',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'cos_group' => 'ANY',
        'avg' => '40272339',
        'min' => '37242156',
        'max' => '43185769',
```

```
          'ninetyfifthpctile' => '43185769'
        },
        {
          'cos_group' => 'expZero',
          'avg' => '1725664',
          'min' => '1610492',
          'max' => '2220659',
          'ninetyfifthpctile' => '2220659'
        }
      ],
      'report_start_time' => '20081020T18:00:00',
      'report_end_time' => '20081020T18:59:59'
    }
}
```

api_traffic_cos_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_cos_ipv4_history {password} {database name} {start time} {end time} {cos} {statistics} {report time range}

This query returns the history of IPv4 CoS (Class of Service) for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **cos—**The Class of Service (CoS) of the traffic.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- cos: string
- end_time: ISO 8601 UTC time
- array of following IPv4 CoS history statistic structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)

— start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $COS = $ARGV[5];
my $StatsType = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos_ipv4_history',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
```

```
                                RPC::XML::RPC_STRING($COS),
                                RPC::XML::RPC_STRING($StatsType),
                                RPC::XML::RPC_STRING($TimeRange))
                                );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '3',
  'network_name' => 'PDlab',
  'report_time' => '20081023T18:54:33',
  'totalEntries' => '3',
  'result' => {
    'history_cos_ipv4' => {
      'end_time' => '20081017T19:00:00',
      'cos' => 'Exp1',
      'statistics' => [
        {
          'average' => '68357',
          'time' => '20081017T17:00:00'
        },
        {
          'average' => '81327',
          'time' => '20081017T18:00:00'
        },
        {
          'average' => '103843',
          'time' => '20081017T19:00:00'
        }
      ],
```

```
        'start_time' => '20081017T17:00:00'
      }
    }
  }
```

api_traffic_cos_vpn

RPC Call: TrafficAnalyzer.api_traffic_cos_vpn {password} {database name} {time} {report time range)

This query returns the traffic statistics for VPN Class of Service (CoS) for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following CoS statistics structure :
  - cos: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
```

```
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs, RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos_vpn',
                              RPC::XML::RPC_STRING($password),
                              RPC::XML::RPC_STRING($database),
                         RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                              RPC::XML::RPC_STRING("hourly"))
                              );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:06:53',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'cos_group' => 'Exp1',
        'avg' => '28579010',
```

```
        'min' => '25433003',
        'max' => '30480646',
        'ninetyfifthpctile' => '30480646'
      },
      {
        'cos_group' => 'expZero',
        'avg' => '11652735',
        'min' => '9744143',
        'max' => '13234953',
        'ninetyfifthpctile' => '13234953'
      }
    ],
    'report_start_time' => '20081020T18:00:00',
    'report_end_time' => '20081020T18:59:59'
  }
}
```

api_traffic_cos_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_cos_vpn_history {history} {database name} {start time} {end time} {cos} {statistics} {report time range}

This query returns the history of VPN CoS (Class of Service) for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **cos—**The Class of Service (CoS) of the traffic whose history is requested.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time

- cos: string
- array of following VPN CoS history statistics structure:
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[2];
my $topoDB = $ARGV[1];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $COS = $ARGV[5];
my $StatsType = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_cos_vpn_history',
```

```
                              RPC::XML::RPC_STRING($PassWord),
                              RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                              RPC::XML::RPC_STRING($COS),
                              RPC::XML::RPC_STRING($StatsType),
                              RPC::XML::RPC_STRING($TimeRange))
                              );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:06:17',
  'totalEntries' => '1',
  'result' => {
    'history_cos_vpn' => {
      'end_time' => '20081015T22:00:00',
      'cos' => 'Exp1',
      'statistics' => [
        {
          'average' => '27691284',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
```

```
        }
    }
}
```

api_traffic_customers_cos_vpn

RPC Call: TrafficAnalyzer.api_traffic_customers_cos_vpn {password} {database name} {time} {report time range}

This query returns the statistics for VPN customers associated with various CoS groups. These statistics are generated from traffic reports for a particular time frame. The statistics include the average, minimum, maximum, and Percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following customer CoS statistics structure :
  - customer_name: string
  - cos_group: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_customers_cos_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

```
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '17',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:59',
  'totalEntries' => '17',
  'result' => {
    'report_result' => [
      {
        'cos_group' => 'Exp1',
        'avg' => '22377892',
        'min' => '17073792',
        'max' => '26069779',
        'ninetyfifthpctile' => '26069779',
        'customer_name' => 'COLA'
      },
      {
        'cos_group' => 'expZero',
        'avg' => '6370741',
        'min' => '4754255',
        'max' => '7866568',
        'ninetyfifthpctile' => '7866568',
        'customer_name' => 'COLA'
      },
      {
        'cos_group' => 'expZero',
        'avg' => '5117454',
        'min' => '4998646',
        'max' => '5397903',
        'ninetyfifthpctile' => '5397903',
        'customer_name' => 'PEPSI'
      },
      {
        'cos_group' => 'Exp1',
        'avg' => '3989614',
```

```
  'min' => '3869208',
  'max' => '4166146',
  'ninetyfifthpctile' => '4166146',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp7',
  'avg' => '3444589',
  'min' => '3303084',
  'max' => '3710184',
  'ninetyfifthpctile' => '3710184',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp6',
  'avg' => '3385884',
  'min' => '3134908',
  'max' => '3641630',
  'ninetyfifthpctile' => '3641630',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp4',
  'avg' => '3245728',
  'min' => '3087837',
  'max' => '3341492',
  'ninetyfifthpctile' => '3341492',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp2',
  'avg' => '3130004',
  'min' => '3014790',
  'max' => '3355601',
  'ninetyfifthpctile' => '3355601',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp5',
  'avg' => '3090363',
  'min' => '2985554',
  'max' => '3327448',
```

```
  'ninetyfifthpctile' => '3327448',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp3',
  'avg' => '3050262',
  'min' => '2964357',
  'max' => '3163878',
  'ninetyfifthpctile' => '3163878',
  'customer_name' => 'PEPSI'
},
{
  'cos_group' => 'Exp1',
  'avg' => '2863454',
  'min' => '0',
  'max' => '6903859',
  'ninetyfifthpctile' => '6903859',
  'customer_name' => 'UNKNOWN'
},
{
  'cos_group' => 'Exp4',
  'avg' => '485037',
  'min' => '392051',
  'max' => '494665',
  'ninetyfifthpctile' => '494665',
  'customer_name' => 'COLA'
},
{
  'cos_group' => 'expZero',
  'avg' => '138565',
  'min' => '0',
  'max' => '966113',
  'ninetyfifthpctile' => '966113',
  'customer_name' => 'UNKNOWN'
},
{
  'cos_group' => 'Exp3',
  'avg' => '118605',
  'min' => '118285',
  'max' => '118784',
  'ninetyfifthpctile' => '118784',
  'customer_name' => 'COLA'
```

```
        },
        {
          'cos_group' => 'Exp6',
          'avg' => '43036',
          'min' => '35794',
          'max' => '54383',
          'ninetyfifthpctile' => '54383',
          'customer_name' => 'COLA'
        },
        {
          'cos_group' => 'Exp4',
          'avg' => '4024',
          'min' => '0',
          'max' => '8048',
          'ninetyfifthpctile' => '8048',
          'customer_name' => 'UNKNOWN'
        },
        {
          'cos_group' => 'Exp6',
          'avg' => '125',
          'min' => '0',
          'max' => '301',
          'ninetyfifthpctile' => '301',
          'customer_name' => 'UNKNOWN'
        }
      ],
      'report_start_time' => '20081020T11:00:00',
      'report_end_time' => '20081020T11:59:59'
    }
  }
```

api_traffic_destination_as_ipv4

RPC Call: TrafficAnalyzer.api_traffic_destination_as_ipv4 {password} {database name} {time} {report time range}

This query returns the IPv4 BGP destination AS statistics from the traffic reports for the requested time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following destination as statistics structure :
  - destination_as: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_destination_as_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

```
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:56',
  'totalEntries' => '5',
  'result' => {
    'report_result' => [
      {
        'avg' => '1155187',
        'min' => '634342',
        'max' => '1263908',
        'ninetyfifthpctile' => '1263908',
        'destination_as' => 'private (64520)'
      },
      {
        'avg' => '874995',
        'min' => '668529',
        'max' => '1027233',
        'ninetyfifthpctile' => '1027233',
        'destination_as' => 'private (64550)'
      },
      {
        'avg' => '442758',
        'min' => '386413',
        'max' => '459441',
        'ninetyfifthpctile' => '459441',
        'destination_as' => 'OSPF (65471)'
      },
      {
        'avg' => '338739',
        'min' => '238202',
        'max' => '357900',
        'ninetyfifthpctile' => '357900',
        'destination_as' => 'private (65400)'
```

```
      },
      {
        'avg' => '147',
        'min' => '0',
        'max' => '215',
        'ninetyfifthpctile' => '215',
        'destination_as' => 'private (65470)'
      }
    ],
    'report_start_time' => '20081020T11:00:00',
    'report_end_time' => '20081020T11:59:59'
  }
}
```

api_traffic_destination_as_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_destination_as_ipv4_history {password} {database} {start time} {end time} {as num} {statistics} {report time range}

This query returns the history of IPv4 destination AS for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **as num—**The AS number whose history needs to be provided.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of the following IPv4 destination AS history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

— destination_as: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[2];
my $topoDB = $ARGV[1];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $AS = $ARGV[5];
my $StatsType = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_destination_as_ipv4_h
istory',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_INT($AS),
                            RPC::XML::RPC_STRING($StatsType),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081027T21:53:22',
  'totalEntries' => '1',
  'result' => {
    'history_destination_as' => {
      'end_time' => '20081015T22:00:00',
      'statistics' => [
        {
          'time' => '20081015T07:00:00',
          'avg' => '1783021'
        }
      ],
      'start_time' => '20081014T17:00:00',
      'destination_as' => '65300'
    }
  }
}
```

api_traffic_egress_pe_vpn

RPC Call: TrafficAnalyzer.api_traffic_egress_pe_vpn {password} {database name} {time} {report time range}

This query returns VPN egress PE router statistics from traffic reports for the requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following egress pe statistics structure :
  - egress_pe: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
```

```
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_egress_pe_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:59',
  'totalEntries' => '6',
  'result' => {
    'report_result' => [
      {
        'avg' => '27187883',
        'min' => '25657362',
        'max' => '28759236',
        'ninetyfifthpctile' => '28759236',
```

```
          'egress_pe' => '10.120.1.6'
        },
        {
          'avg' => '18010082',
          'min' => '15742764',
          'max' => '18810278',
          'ninetyfifthpctile' => '18810278',
          'egress_pe' => '10.120.1.8'
        },
        {
          'avg' => '10499123',
          'min' => '10162353',
          'max' => '11873623',
          'ninetyfifthpctile' => '11873623',
          'egress_pe' => '10.120.1.9'
        },
        {
          'avg' => '3724195',
          'min' => '3272774',
          'max' => '3867974',
          'ninetyfifthpctile' => '3867974',
          'egress_pe' => '10.120.1.14'
        },
        {
          'avg' => '177531',
          'min' => '164544',
          'max' => '191998',
          'ninetyfifthpctile' => '191998',
          'egress_pe' => '10.120.1.7'
        },
        {
          'avg' => '74',
          'min' => '0',
          'max' => '149',
          'ninetyfifthpctile' => '149',
          'egress_pe' => '10.120.1.5'
        }
      ],
      'report_start_time' => '20081020T11:00:00',
      'report_end_time' => '20081020T11:59:59'
    }
  }
```

api_traffic_egress_pe_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_egress_pe_vpn_history {password} {database} {start time} {end time} {egress rtr} {statistics} {report time range}

This query returns the history of VPN egress PE's for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.
- **egress rtr**—The egress router's IP address whose history needs to be provided.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of following VPN egress PE history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - egress_pe: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
```

```
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_egress_pe_vpn_history
',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
```

```
  },
  'numReturnedEntries' => '3',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:24:49',
  'totalEntries' => '3',
  'result' => {
    'history_egress_pe_vpn' => {
      'end_time' => '20081017T20:00:00',
      'statistics' => [
        {
          'percentile' => '216763',
          'time' => '20081017T18:00:00'
        },
        {
          'percentile' => '194747',
          'time' => '20081017T19:00:00'
        },
        {
          'percentile' => '193763',
          'time' => '20081017T20:00:00'
        }
      ],
      'start_time' => '20081017T18:00:00',
      'egress_pe' => 'DC-PE1-ROUTER7'
    }
  }
}
```

api_traffic_egress_router_ipv4

RPC Call: TrafficAnalyzer.api_traffic_egress_router_ipv4 {password} {database name} {time} {report time range}

This query returns IPv4 BGP egress router statistics from traffic reports for the requested time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following egress router statistics structure :

- egress_router: string
- next_hop: string
- avg: double (bps)
- min: double (bps)
- max: double (bps)
- ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_egress_router_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:56',
  'totalEntries' => '6',
  'result' => {
    'report_result' => [
      {
        'next_hop' => '10.71.6.29',
        'avg' => '2472940',
        'min' => '1750429',
        'egress_router' => '10.120.1.5',
        'max' => '2742540',
        'ninetyfifthpctile' => '2742540'
      },
      {
        'next_hop' => '10.64.15.5',
        'avg' => '527818',
        'min' => '507434',
        'egress_router' => '10.120.1.5',
        'max' => '580476',
        'ninetyfifthpctile' => '580476'
      },
```

```
      {
        'next_hop' => '10.120.1.1',
        'avg' => '421642',
        'min' => '342939',
        'egress_router' => '10.120.1.1',
        'max' => '429978',
        'ninetyfifthpctile' => '429978'
      },
      {
        'next_hop' => '10.73.6.5',
        'avg' => '338739',
        'min' => '238202',
        'egress_router' => '10.120.1.1',
        'max' => '357900',
        'ninetyfifthpctile' => '357900'
      },
      {
        'next_hop' => '10.72.6.42',
        'avg' => '147',
        'min' => '0',
        'egress_router' => '10.120.1.4',
        'max' => '215',
        'ninetyfifthpctile' => '215'
      },
      {
        'next_hop' => '192.168.0.254',
        'avg' => '44',
        'min' => '27',
        'egress_router' => '10.120.1.5',
        'max' => '115',
        'ninetyfifthpctile' => '115'
      }
    ],
    'report_start_time' => '20081020T11:00:00',
    'report_end_time' => '20081020T11:59:59'
  }
}
```

api_traffic_egress_router_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_egress_router_ipv4_history {password} {database} {start time} {end time} {egress rtr} {next hop} {statistics} {report time range}

This query returns history of IPv4 egress routers for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **egress rtr**—The egress router's IP address whose history needs to be provided.
- **next hop**—The IP address of the next hop which, combined with the egress router, identifies the element whose history needs to be retrieved.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string

- end_time: ISO 8601 UTC time
- next_hop: string
- array of following IPv4 egress router history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exit_router_ip: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $ExitRtrIP = $ARGV[5];
my $NextHop = $ARGV[6];
my $TypeOfStatistics = $ARGV[7];
my $TimeRange = $ARGV[8];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_egress_router_ipv4_hi
story',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($ExitRtrIP),
                            RPC::XML::RPC_STRING($NextHop),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:12:09',
  'totalEntries' => '2',
  'result' => {
    'history_egress_router' => {
      'end_time' => '20081014T18:00:00',
      'next_hop' => '10.73.6.5',
      'statistics' => [
        {
          'time' => '20081014T17:00:00',
          'avg' => '3275795'
```

```
          },
          {
            'time' => '20081014T18:00:00',
            'avg' => '3405443'
          }
        ],
        'start_time' => '20081014T17:00:00',
        'exit_router_ip' => '10.120.1.1'
      }
    }
  }
```

api_traffic_exporters

RPC Call: TrafficAnalyzer.api_traffic_exporters {password} {database name} {time} {report time range}

This query returns traffic statistics for aggregate exporters for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.

- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following exporter statistics structure :
  - exporter: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
```

```
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
```

```
'numReturnedEntries' => '4',
'network_name' => 'PDlab',
'report_time' => '20081021T03:08:00',
'totalEntries' => '4',
'result' => {
  'report_result' => [
    {
      'avg' => '41763537',
      'min' => '39367271',
      'max' => '45839141',
      'ninetyfifthpctile' => '45839141',
      'exporter' => '10.120.1.1'
    },
    {
      'avg' => '21745267',
      'min' => '21693625',
      'max' => '21815174',
      'ninetyfifthpctile' => '21815174',
      'exporter' => '10.120.1.2'
    },
    {
      'avg' => '20197343',
      'min' => '19375645',
      'max' => '23505399',
      'ninetyfifthpctile' => '23505399',
      'exporter' => '10.120.1.4'
    },
    {
      'avg' => '17550175',
      'min' => '16526164',
      'max' => '18760502',
      'ninetyfifthpctile' => '18760502',
      'exporter' => '10.120.1.3'
    }
  ],
  'report_start_time' => '20081020T11:00:00',
  'report_end_time' => '20081020T11:59:59'
}
```

api_traffic_exporters_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_history {password} {database} {start time} {end time} {exporter} {statistics} {report time range}

This query returns the history of aggregate exporters for the requested time period. It will retrieve the history statistics of the average, minimum, maximum, or percentile.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history needs to be provided. This can be a router name or IP address.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of following aggregate exporters history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
```

```
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_history',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
```

```
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081027T21:05:00',
  'totalEntries' => '1',
  'result' => {
    'history_exporters' => {
      'end_time' => '20081021T12:00:00',
      'statistics' => [
        {
          'time' => '20081021T07:00:00',
          'avg' => '10368741'
        }
      ],
      'start_time' => '20081020T12:00:00',
      'exporter' => 'SF-CORE-ROUTER2'
    }
  }
}
```

api_traffic_exporters_idx

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx {password} {database name} {time} {report time range}

This query returns statistics from traffic reports for aggregate exporter interfaces for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following exporter interface statistics structure :
  - source: string
  - destination: string
  - dst_interface: struct
    - address: string
    - name: string
    - capacity: string
    - description: string
    - index: string
  - min: double (bps)
  - max: double (bps)
  - avg: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
```

```
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_idx',
                           RPC::XML::RPC_STRING($password),
                           RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                           RPC::XML::RPC_STRING("hourly"))
                           );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => '9.3.17'
  },
  'numReturnedEntries' => '40',
  'network_name' => 'PDIHQ',
  'network_time' => '20110907T17:00:00',
  'report_time' => '20110912T18:40:25',
  'totalEntries' => '40',
  'result' => {
    'report_result' => [
      {
        'source' => 'LA-P-7204-R1.04',
        'destination' => 'LA-P-7204-R1',
        'avg' => '75305529',
        'min' => '--',
        'interface' => {
          'index' => '8',
          'name' => 'Gi0/3',
          'address' => '10.74.5.1/24',
          'capacity' => '100.00 Mbps',
          'description' => 'Gi0/3_10.74.5.1_R1_To_R3'
},
        'max' => '--',
        'ninetyfifthpctile' => '--'
      },
      {
        'source' => 'LA-PE-ERX-R17.02',
        'destination' => 'LA-P-7204-R3',
        'avg' => '54346337',
        'min' => '--',
        'interface' => {
          'index' => '5',
          'name' => 'Gi0/3',
          'address' => '10.64.6.3/24',
          'capacity' => '100.00 Mbps',
          'description' => 'GigabitEthernet0/3_R3-R17-IxNet150'
        },
        'max' => '--',
```

```
          'ninetyfifthpctile' => '--'
        }
       ],
      'report_start_time' => '20110814T07:00:00',
      'report_end_time' => '20110821T06:59:59'
    }
}
```

api_traffic_exporters_idx_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx_history {password} {database} {start time} {end time} {exporter} {index} {statistics} {report time range}

This query returns the history of aggregate exporter interfaces for the requested time period. It will retrieve the history of the average, minimum, maximum or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history needs to be provided. This can be a router name or IP address.
- **index**—The index for the exporter whose history needs to be retrieved.

- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- index: string
- array of following aggregate exporter interfaces history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
```

```
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_history',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
```

```
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '25',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:29:46',
  'totalEntries' => '25',
  'result' => {
    'history_exporters' => {
      'end_time' => '20081017T19:00:00',
      'statistics' => [
        {
          'time' => '20081017T17:00:00',
          'avg' => '42074897'
        },
        {
          'time' => '20081017T17:05:00',
          'avg' => '39418329'
        },
        {
          'time' => '20081017T17:10:00',
          'avg' => '37859057'
        },
        {
          'time' => '20081017T17:15:00',
          'avg' => '43451337'
        }
      ],
      'start_time' => '20081017T17:00:00',
      'exporter' => 'SF-CORE-RTR1'
    }
  }
}
```

api_traffic_exporters_idx_ipv4

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx_ipv4 {password} {database name} {time} {report time range}

This query returns the statistics from traffic reports for IPv4 exporters with interfaces for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int

- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following exporter interfaces statistics structure :
  - source: string
  - destination: string
  - dst_interface: struct
    - address: string
    - name: string
    - capacity: string
    - description: string
    - index: string
  - min: double (bps)
  - max: double (bps)
  - avg: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
```

```
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_idx_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '4',
  'network_name' => 'PDlab',
  'report_time' => '20081021T21:17:48',
  'totalEntries' => '4',
  'result' => {
    'report_result' => [
      {
        'index' => '2',
        'avg' => '11080529',
        'min' => '0',
```

```
          'max' => '112758205',
          'ninetyfifthpctile' => '11082399',
          'exporter' => '10.120.1.2'
        },
        {
          'index' => '5',
          'avg' => '10952687',
          'min' => '0',
          'max' => '54792654',
          'ninetyfifthpctile' => '28757911',
          'exporter' => '10.120.1.1'
        },
        {
          'index' => '1',
          'avg' => '4311266',
          'min' => '0',
          'max' => '20666080',
          'ninetyfifthpctile' => '10811811',
          'exporter' => '10.120.1.4'
        },
        {
          'index' => '2',
          'avg' => '1757309',
          'min' => '0',
          'max' => '9946306',
          'ninetyfifthpctile' => '4679360',
          'exporter' => '10.120.1.3'
        }
      ],
      'report_start_time' => '20080901T07:00:00',
      'report_end_time' => '20081001T06:59:59'
    }
  }
```

api_traffic_exporters_idx_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx_ipv4_history {password} {database name} {start time} {end time} {exporter} {index} {statistics} {report time range}

This query returns the history of IPv4 exporter interfaces for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history needs to be provided. This can be a router name or IP address.
- **index**—The index for the exporter whose history needs to be retrieved.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string

- end_time: ISO 8601 UTC time
- index: string
- array of following IPv4 exporters interfaces history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}


my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $IntfIndex = $ARGV[6];
my $TypeOfStatistics = $ARGV[7];
my $TimeRange = $ARGV[8];



use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";
```

```
my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_idx_ipv4_hi
story',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($IntfIndex),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081027T21:16:11',
  'totalEntries' => '1',
  'result' => {
    'history_exporters_idx_ipv4' => {
      'end_time' => '20081021T12:00:00',
```

```
        'index' => '1',
        'statistics' => [
          {
            'time' => '20081021T07:00:00',
            'avg' => '491546'
          }
        ],
        'start_time' => '20081020T12:00:00',
        'exporter' => 'SF-CORE-ROUTER2'
      }
    }
}
```

api_traffic_exporters_idx_vpn

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx_vpn {password} {database name} {time} {report time range}

This query returns statistics from traffic reports for VPN exporter interfaces for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following exporter interface statistics structure :
  - source: string
  - destination: string
  - dst_interface: struct
    - address: string
    - name: string
    - capacity: string
    - description: string
    - index: string
  - min: double (bps)
  - max: double (bps)
  - avg: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
```

```
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_idx_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
```

```
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '4',
  'network_name' => 'PDlab',
  'report_time' => '20081021T21:20:45',
  'totalEntries' => '4',
  'result' => [
    {
      'index' => '1',
      'avg' => '10147560',
      'min' => '0',
      'max' => '53254917',
      'ninetyfifthpctile' => '26293146',
      'exporter' => '10.120.1.4'
    },
    {
      'index' => '1',
      'avg' => '6017612',
      'min' => '0',
      'max' => '33400901',
      'ninetyfifthpctile' => '15136558',
      'exporter' => '10.120.1.2'
    },
    {
      'index' => '3',
      'avg' => '3437543',
      'min' => '0',
      'max' => '16642681',
      'ninetyfifthpctile' => '8140292',
      'exporter' => '10.120.1.3'
    },
    {
      'index' => '4',
      'avg' => '1620785',
      'min' => '0',
      'max' => '7724461',
      'ninetyfifthpctile' => '3105202',
      'exporter' => '10.120.1.1'
    }
     ],
    'report_start_time' => '20080901T07:00:00',
```

```
    'report_end_time' => '20081001T06:59:59'
  }
}
```

api_traffic_exporters_idx_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_idx_vpn_history {password} {database} {start time} {end time} {exporter} {index} {statistics} {report time range}

This query returns the history of VPN exporter interfaces for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history needs to be provided. This can be a router name or IP address.
- **index**—The index for the exporter whose history needs to be retrieved.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct

- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- index: string
- array of following VPN exporters interfaces history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}


my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $IntfIndex = $ARGV[6];
my $TypeOfStatistics = $ARGV[7];
my $TimeRange = $ARGV[8];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
```

```
my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_idx_vpn_his
tory',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($IntfIndex),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
```

```
  'report_time' => '20081023T20:49:08',
  'totalEntries' => '2',
  'result' => {
    'history_exporters_idx_vpn' => {
      'end_time' => '20081015T18:00:00',
      'index' => '1',
      'statistics' => [
        {
          'time' => '20081014T07:00:00',
          'max' => '10487650'
        },
        {
          'time' => '20081015T07:00:00',
          'max' => '10499684'
        }
      ],
      'start_time' => '20081013T17:00:00',
      'exporter' => 'SF-CORE-ROUTER2'
    }
  }
}
```

api_traffic_exporters_ipv4

RPC Call: TrafficAnalyzer.api_traffic_exporters_ipv4 {password} {database name} {time} {report time range}

This query returns traffic statistics for IPv4 exporters for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following IPv4 exporter statistics structure :
  - exporter: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)

— ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
```

```
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '4',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:54',
  'totalEntries' => '4',
  'result' => {
    'report_result' => [
      {
        'avg' => '26965755',
        'min' => '24932603',
        'max' => '29656154',
        'ninetyfifthpctile' => '29656154',
        'exporter' => '10.120.1.1'
      },
      {
        'avg' => '11313584',
        'min' => '11283964',
        'max' => '11341489',
        'ninetyfifthpctile' => '11341489',
        'exporter' => '10.120.1.2'
      },
      {
        'avg' => '2393995',
        'min' => '1873828',
        'max' => '4089243',
        'ninetyfifthpctile' => '4089243',
        'exporter' => '10.120.1.4'
      },
      {
        'avg' => '981851',
        'min' => '835986',
```

```
          'max' => '1015164',
          'ninetyfifthpctile' => '1015164',
          'exporter' => '10.120.1.3'
        }
      ],
      'report_start_time' => '20081020T11:00:00',
      'report_end_time' => '20081020T11:59:59'
    }
}
```

api_traffic_exporters_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_ipv4_history {password} {database} {start time} {end time} {exporter} {statistics} report time range}

This query returns the history of IPv4 exporters for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history is provided. This can be a router name or IP address.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of the following IPv4 exporter history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)

- start_time: ISO 8601 UTC time
- exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_ipv4_histor
y',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '4',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:29:46',
  'totalEntries' => '4',
  'result' => {
    'history_exporters_ipv4' => {
      'end_time' => '20081017T19:00:00',
      'statistics' => [
        {
          'time' => '20081017T17:00:00',
          'avg' => '42074897'
        },
        {
          'time' => '20081017T17:05:00',
          'avg' => '39418329'
        },
        {
          'time' => '20081017T17:10:00',
          'avg' => '37859057'
        },
```

```
          {
            'time' => '20081017T17:15:00',
            'avg' => '43451337'
          }
        ],
        'start_time' => '20081017T17:00:00',
        'exporter' => 'SF-CORE-RTR1'
      }
    }
}
```

api_traffic_exporters_vpn

RPC Call: TrafficAnalyzer.api_traffic_exporters_vpn {password} {database name} {time} {report time range}

This query returns traffic statistics for VPN exporters for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.

- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following exporter statistics structure :
  - exporter: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
```

```
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '4',
```

```
'network_name' => 'PDlab',
'report_time' => '20081021T03:07:58',
'totalEntries' => '4',
'result' => {
  'report_result' => [
    {
      'avg' => '17803348',
      'min' => '16907540',
      'max' => '19416156',
      'ninetyfifthpctile' => '19416156',
      'exporter' => '10.120.1.4'
    },
    {
      'avg' => '16568323',
      'min' => '15539384',
      'max' => '17745895',
      'ninetyfifthpctile' => '17745895',
      'exporter' => '10.120.1.3'
    },
    {
      'avg' => '14797781',
      'min' => '13047579',
      'max' => '16243283',
      'ninetyfifthpctile' => '16243283',
      'exporter' => '10.120.1.1'
    },
    {
      'avg' => '10431682',
      'min' => '10379332',
      'max' => '10483355',
      'ninetyfifthpctile' => '10483355',
      'exporter' => '10.120.1.2'
    }
  ],
  'report_start_time' => '20081020T11:00:00',
  'report_end_time' => '20081020T11:59:59'
}
```

api_traffic_exporters_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_exporters_vpn_history {password} {database} {start time} {end time} {exporter} {statistics} {report time range}

This query returns the history of VPN exporters for the requested time period. It will return the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **exporter**—The exporter whose history is provided. This can be a router name or IP address.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of the following VPN exporters history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - exporter: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
```

```
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_exporters_vpn_history
',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
```

```
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081027T21:05:00',
  'totalEntries' => '1',
  'result' => {
    'history_exporters_vpn' => {
      'end_time' => '20081021T12:00:00',
      'statistics' => [
        {
          'time' => '20081021T07:00:00',
          'avg' => '10368741'
        }
      ],
      'start_time' => '20081020T12:00:00',
      'exporter' => 'SF-CORE-ROUTER2'
    }
  }
}
```

api_traffic_ingress_pe_vpn

RPC Call: TrafficAnalyzer.api_traffic_ingress_pe_vpn {password} {database name} {time} {report time range}

This query returns the VPN Ingress PE router statistics from traffic reports for the requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following ingress PE statistics structure :
  - ingress_pe: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);
```

```
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_ingress_pe_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '6',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:59',
  'totalEntries' => '6',
```

```
'result' => {
  'report_result' => [
    {
      'avg' => '29217756',
      'min' => '22544041',
      'max' => '32558487',
      'ninetyfifthpctile' => '32558487',
      'ingress_pe' => '10.120.1.7'
    },
    {
      'avg' => '17516627',
      'min' => '15342098',
      'max' => '18307461',
      'ninetyfifthpctile' => '18307461',
      'ingress_pe' => '10.120.1.9'
    },
    {
      'avg' => '10928856',
      'min' => '10568499',
      'max' => '12309425',
      'ninetyfifthpctile' => '12309425',
      'ingress_pe' => '10.120.1.8'
    },
    {
      'avg' => '2343151',
      'min' => '0',
      'max' => '6947605',
      'ninetyfifthpctile' => '6947605',
      'ingress_pe' => 'Unknown'
    },
    {
      'avg' => '162698',
      'min' => '120218',
      'max' => '181131',
      'ninetyfifthpctile' => '181131',
      'ingress_pe' => '10.120.1.6'
    },
    {
      'avg' => '17834',
      'min' => '16285',
      'max' => '19532',
      'ninetyfifthpctile' => '19532',
```

```
          'ingress_pe' => '10.120.1.14'
        }
      ],
      'report_start_time' => '20081020T11:00:00',
      'report_end_time' => '20081020T11:59:59'
    }
  }
```

api_traffic_ingress_pe_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_ingress_pe_vpn_history {password} {database} {start time} {end time} {ingress rtr} {statistics} {report time range}

This query returns the history of VPN Ingress PE for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.

- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **ingress rtr**—The ingress PE router name or IP address whose history needs to be provided.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of following VPN egress pe history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - ingress_pe: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
```

```
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $RouterName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_ingress_pe_vpn_histor
y',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($RouterName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

```
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '5',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:22:01',
  'totalEntries' => '5',
  'result' => {
    'history_ingress_pe_vpn' => {
      'end_time' => '20081017T20:00:00',
      'statistics' => [
        {
          'time' => '20081017T18:00:00',
          'avg' => '25055694'
        },
        {
          'time' => '20081017T18:05:00',
          'avg' => '30104859'
        },
        {
          'time' => '20081017T18:10:00',
          'avg' => '27357238'
        },
        {
          'time' => '20081017T18:15:00',
          'avg' => '24317577'
        },
        {
          'time' => '20081017T18:20:00',
          'avg' => '30557394'
        }
      ],
      'start_time' => '20081017T18:00:00',
      'ingress_pe' => 'DC-PE1-ROUTER7'
    }
  }
```

}

api_traffic_flow_records

RPC Call: TrafficAnalyzer.api_traffic_flow_records {password} {database name} {start time} {end time} {filter}

This API returns the flow records for a given time period specified by start time and end time. It can be filtered by exporter IP address and input SNMP index. This API must be used in conjunction with the api_mp_list_handle and api_mp_close_handle calls as described in Chapter 3, "Using Re-Entrant Queries"

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which should be the top-level administrative domain for the network, such as CorpNet.
- **start_time**—A time specified in ISO8601 format in the UTC time zone, such as 20050725T21:47:35. This is the start time for which the flow records is queried.
- **end_time**—A time specified in ISO8601 format in the UTC time zone, such as 20050725T21:47:35. This is the end time for which the flow records is queried.
- **filter**—String representing a filter entry, which can be any of the following. All the filters can be combined using and/or/not operators.
  - ipv4SrcAddr ipAddress (example 10.120.1.1)
  - ipv4DstAddr ipAddress (example 10.120.1.1)
  - ingressInterfaceIndex integer (example. 3)
  - rfExporterIP ipAddress (example 10.120.1.1)
  - rfTrafficGroup string (example TG1)
  - rfCoSGroup string (example CG1)
  - rfEgressPE ipAddress (example 10.120.1.1)

- rfAddrFamily string (IPV4 / VPN / VPN_TO_INET / INET_TO_VPN)
- rfTunnelName string (Tunnel1)
- rfFRRName string (FRR1)
- rfEgressVRF string (vpn3)

Structure of Output

- start_time: ISO 8601 UTC time
- end_time: ISO 8601 UTC time
- exporter_ip: string
- exporter_interface_index: integer
- source_addr: string
- destination_address: string
- source_port_name: string
- source_port_no: int
- destination_port_name: string
- destination_port_no: int
- protocol: string
- ip_tos: integer
- bytes: string
- outgoing_interface_index: integer
- label 1: string
- label 2: string
- label 3: string
- mpls_top_label_FEC: string
- mpls_top_label_type: string
- direction: integer
- sampling_adj_bytes: integer
- packets: string
- samplsing_adjusted_packets: integer

- all_flows_loaded: boolean value
  - 0 - all flows have not yet been loaded
  - 1 - all flows have been loaded
- traffic_group: string
- cos_group: string
- tunnel: string
- frr: string
- vrf_name: string
- address_family: string

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: TrafficAnalyzer.api_traffic_flow_records ip database
[filter]\n";
    exit(0);
}

my $rexip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $password = 'admin';
my $exporter = '10.120.1.8';
my $idx = 3;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = str2time("11 Apr 2011 11:00:00 PST");
```

```
my $t2 = str2time("11 Apr 2011 13:00:00 PST");

my $openreq = RPC::XML::request->new(
                'TrafficAnalyzer.api_traffic_flow_records',
                RPC::XML::RPC_STRING($password),
                RPC::XML::RPC_STRING($database),
                RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                RPC::XML::datetime_iso8601->new(time2iso8601($t2)),
                RPC::XML::RPC_STRING($exporter),
                RPC::XML::RPC_INT($idx)
                );

my $openres = $client->send_request($openreq);
if ($openres->is_fault) {print("---XMLRPC FAULT ---"); }
my $overall = $openres->value;

my $handle = int($overall->{result});

my $index = 0;
my $step = 10000;
my $result;
my $done = '0';
while($done == 0) {
my $listreq = RPC::XML::request->new(
                                'RouteAnalyzer.api_mp_list_handle',
                                RPC::XML::RPC_INT($handle),
                                RPC::XML::RPC_STRING($database),
                                RPC::XML::RPC_INT($index),
                                RPC::XML::RPC_INT($step),
                                );
my $listresp = $client->send_request($listreq);
     push @{$result}, $listresp->value;
     $done = $listresp->value->{all_flows_loaded};
}

my $p = Dumper($result);
print $p;


my $closereq = RPC::XML::request->new(
                'RouteAnalyzer.api_mp_close_handle',
                RPC::XML::RPC_INT($handle),
```

```
                RPC::XML::RPC_STRING($database),
);
my $closeres = $client->send_request($closereq);
```

Sample Output

```
[
  {
    'vinfo' => {
      'software_version' => '2011.10.07.14.24-E Traffic Explorer',
      'appliance_version' => '9.4.12'
    },
    'end_time' => '20111008T19:05:00',
    'numReturnedEntries' => '50250',
    'start_time' => '20111008T19:00:00',
    'flows_records' => [
      {
        'protocol' => 'tcp',
        'bytes' => '40.00',
        'traffic_group' => 'Other',
        'packets' => '1.00',
        'source_addr' => '10.71.2.32',
        'address_family' => 'IPv4',
        'mpls_top_label_type' => '-1',
        'exporter_interface_idx' => '1',
        'sampling_adj_bytes' => '40.00',
        'direction' => '-1',
        'outgoing_interface_idx' => 'Not Available',
        'end_time' => '20111008T19:04:44',
        'vrf_name' => '',
        'cos_group' => 'ANY',
        'label3' => 'Not Available',
        'destination_port_no' => '12308',
        'label2' => 'Not Available',
        'frr' => '',
        'source_port_name' => 'bgp',
        'sampling_adj_packets' => '1.00',
        'label1' => 'Not Available',
        'start_time_' => '20111008T19:04:44',
        'mpls_top_label_FEC' => 'Not Available',
        'destination_addr' => '10.71.3.28',
        'source_port_no' => '179',
```

```
          'exporter_ip' => '10.130.1.28',
          'tunnel' => '',
          'destination_port_name' => '12308',
          'ip_tos' => '192'
        },
        {
          'protocol' => 'tcp',
          'bytes' => '59.00',
          'traffic_group' => 'Other',
          'packets' => '1.00',
          'source_addr' => '10.71.2.32',
          'address_family' => 'IPv4',
          'mpls_top_label_type' => '-1',
          'exporter_interface_idx' => '1',
          'sampling_adj_bytes' => '59.00',
          'direction' => '-1',
          'outgoing_interface_idx' => 'Not Available',
          'end_time' => '20111008T19:04:11',
          'vrf_name' => '',
          'cos_group' => 'ANY',
          'label3' => 'Not Available',
          'destination_port_no' => '12308',
          'label2' => 'Not Available',
          'frr' => '',
          'source_port_name' => 'bgp',
          'sampling_adj_packets' => '1.00',
          'label1' => 'Not Available',
          'start_time_' => '20111008T19:04:11',
          'mpls_top_label_FEC' => 'Not Available',
          'destination_addr' => '10.71.3.28',
          'source_port_no' => '179',
          'exporter_ip' => '10.130.1.28',
          'tunnel' => '',
          'destination_port_name' => '12308',
          'ip_tos' => '192'
        }
        ],
      'all_flows_loaded' => 1
    }
]
```

api_traffic_links

RPC Call: TrafficAnalyzer.api_traffic_links {password} {database name} {time} {report time range}

This query returns the traffic statistics for aggregate links for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: verstion struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time

- array of the following link statistics structures:
  - source: string
  - destination: string
  - src_intf_address: string
  - dst_intf_address: string
  - intf_index: int
  - intf_name: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
- ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs, RPC::XML::request->new('TrafficAnalyzer.api_traffic_links',
```

```
                              RPC::XML::RPC_STRING($password),
                              RPC::XML::RPC_STRING($database),
                         RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                              RPC::XML::RPC_STRING("hourly"))
                              );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}



 }
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '2011.04.26.17.33-E Traffic Explorer',
    'appliance_version' => '9.2.25'
  },
  'numReturnedEntries' => '3',
  'network_name' => 'PDIHQ',
  'network_time' => '20110426T19:55:17',
  'report_time' => '20110427T00:50:12',
  'totalEntries' => '3',
  'result' => {
        'report_result' => [
      {
             'source' => 'DC-P-7204-R13',
            'src_intf_address' => '10.64.26.13/24',
            'destination' => 'DC-P-7204-R13.03',
            'avg' => '6502858',
            'min' => '--',
            'max' => '--',
            'dst_intf_address' => '--',
            'intf_index' => '2',
             'intf_name' => 'Fa1/0',
```

```
            'ninetyfifthpctile' => '--'
          },
        {
          'source' => 'SJ-P-7204-R2',
          'src_intf_address' => '10.64.3.2/24',
          'destination' => 'SJ-P-7204-R2.EF',
          'avg' => '23',
          'min' => '--',
          'max' => '--',
          'dst_intf_address' => '--',
          'intf_index' => '5',
          'intf_name' => 'Gi0/1',
          'ninetyfifthpctile' => '--'
        },
        {
          'source' => 'SJ-PE-2811-R8',
          'src_intf_address' => '10.64.11.8/24',
          'destination' => 'SJ-PE-2811-R8.02',
          'avg' => '23',
          'min' => '--',
          'max' => '--',
          'dst_intf_address' => '--',
          'intf_index' => '2',
          'intf_name' => 'Fa0/0',
          'ninetyfifthpctile' => '--'
        }
      ],
      'report_start_time' => '20110426T19:50:00',
      'report_end_time' => '20110426T19:54:59'
    }
  }
```

api_traffic_links_cos

RPC Call: TrafficAnalyzer.api_traffic_links_cos {password} {database name} {time} {report time range}

This query returns the traffic statistics for aggregate links for various Class of Service (CoS) groups for a particular time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report _start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following CoS statistics structures :
  - source: string
  - cos: string
  - destination: string

- avg: double (bps)
- min: double (bps)
- max: double (bps)
- ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
```

```
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:17:27',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'source' => 'SF-CORE-RTR1',
        'cos_group' => 'Exp1',
        'destination' => 'SF-CORE-RTR1.03',
        'avg' => '24990988',
        'min' => '21761109',
        'max' => '26883733',
        'ninetyfifthpctile' => '26883733'
      },
      {
        'source' => 'SF-CORE-RTR1.03',
        'cos_group' => 'Exp1',
        'destination' => 'SF-PE1-ROUTER6',
        'avg' => '24990988',
        'min' => '21761109',
        'max' => '26883733',
        'ninetyfifthpctile' => '26883733'
      }
    ],
    'report_start_time' => '20081020T18:00:00',
    'report_end_time' => '20081020T18:59:59'
  }
}
```

api_traffic_links_cos_history

RPC Call: TrafficAnalyzer.api_traffic_links_cos_history {password} {database name} {start time} {end time} {source} {destination} {cos} {statistics} {report time range}

This query returns the history of the aggregate links for various Class of Service (CoS). It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.

- **source**—The source of the aggregate link for a CoS whose history needs to be provided. This can be the router name or the IP address of the router.
- **destination**—The destination of the aggregate link for a CoS whose history needs to be provided. This can be the router name or the IP address of the router.
- **cos**—The Class of Service (CoS) for the traffic.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- destination: string
- cos: string
- array of following aggregate CoS links history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
```

```
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[2];
my $topoDB = $ARGV[1];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $SRlink = $ARGV[5];
my $Dstlink = $ARGV[6];
my $COS = $ARGV[7];
my $StatsType = $ARGV[8];
my $TimeRange = $ARGV[9];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos_history',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($SRlink),
                            RPC::XML::RPC_STRING($Dstlink),
                            RPC::XML::RPC_STRING($COS),
                            RPC::XML::RPC_STRING($StatsType),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
```

```
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:56:41',
  'totalEntries' => '1',
  'result' => {
    'history_links_cos' => {
      'source' => 'SF-CORE-ROUTER2',
      'end_time' => '20081015T22:00:00',
      'cos' => 'Exp1',
      'destination' => 'SF-PE2-ROUTER8.01',
      'statistics' => [
        {
          'average' => '2455007',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_links_cos_ipv4

RPC Call: TrafficAnalyzer.api_traffic_links_cos_ipv4 {password} {database name} {time} {report time range}

This query returns the traffic statistics for IPv4 links for various CoS (class of service) groups, at a particular time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time

- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of link CoS statistics containing the following structure :
  - source: string
  - cos: string
  - destination: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:12:09',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'source' => 'SF-CORE-RTR1',
        'cos_group' => 'ANY',
        'destination' => 'CORE-ROUTER13.03',
        'avg' => '12174077',
        'min' => '9074353',
        'max' => '13809400',
        'ninetyfifthpctile' => '13809400'
      },
      {
        'source' => 'CORE-ROUTER13.03',
        'cos_group' => 'ANY',
        'destination' => 'CORE-ROUTER13',
```

```
          'avg' => '12174077',
          'min' => '9074353',
          'max' => '13809400',
          'ninetyfifthpctile' => '13809400'
        }
      ],
      'report_start_time' => '20081020T18:00:00',
      'report_end_time' => '20081020T18:59:59'
    }
  }
```

api_traffic_links_cos_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_links_cos_ipv4_history {password} {database name} {start time} {end time} {source} {destination} {cos} {statistics} {report time range}

This query returns the history for IPv4 links for various Class of Services (CoS). It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.

- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **source**—The source of the IPv4 link whose history needs to be provided. This can be the router name or the IP address of the router.
- **destination**—The destination of the IPv4 link whose history needs to be provided. This can be the router name or the IP address of the router.
- **cos**—The Class of Service (CoS) for the traffic.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- cos: string
- destination: string
- array of following IPv4 links CoS history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)

— start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[2];
my $topoDB = $ARGV[1];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $SRlink = $ARGV[5];
my $Dstlink = $ARGV[6];
my $COS = $ARGV[7];
my $StatsType = $ARGV[8];
my $TimeRange = $ARGV[9];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos_ipv4_histor
y',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($SRlink),
                            RPC::XML::RPC_STRING($Dstlink),
                            RPC::XML::RPC_STRING($COS),
                            RPC::XML::RPC_STRING($StatsType),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:50:39',
  'totalEntries' => '1',
  'result' => {
    'history_links_cos_ipv4' => {
      'source' => 'SF-CORE-ROUTER2',
      'end_time' => '20081015T22:00:00',
      'cos' => 'Exp1',
      'destination' => 'SF-PE2-ROUTER8.01',
      'statistics' => [
        {
          'average' => '13221',
          'time' => '20081015T07:00:00'
        }
      ],
```

```
        'start_time' => '20081014T17:00:00'
      }
    }
  }
```

api_traffic_links_cos_vpn

RPC Call: TrafficAnalyzer.api_traffic_links_cos_vpn {password} {database name} {time} {report time range}

This query returns the traffic statistics for VPN links for various CoS (Class of Service) groups for a particular time range. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following link CoS statistics structures :
  - source: string
  - cos: string
  - destination: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
```

```
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T00:16:10',
  'totalEntries' => '2',
  'result' => {
```

```
    'report_result' => [
      {
        'source' => 'SF-CORE-RTR1',
        'cos_group' => 'Exp1',
        'destination' => 'SF-CORE-RTR1.03',
        'avg' => '24990988',
        'min' => '21761109',
        'max' => '26883733',
        'ninetyfifthpctile' => '26883733'
      },
      {
        'source' => 'SF-CORE-RTR1.03',
        'cos_group' => 'Exp1',
        'destination' => 'SF-PE1-ROUTER6',
        'avg' => '24990988',
        'min' => '21761109',
        'max' => '26883733',
        'ninetyfifthpctile' => '26883733'
      }
    ],
    'report_start_time' => '20081020T18:00:00',
    'report_end_time' => '20081020T18:59:59'
  }
}
```

api_traffic_links_cos_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_links_cos_vpn_history {password} {database name} {start time} {end time} {source} {destination} {cos} {statistics} {report time range}

This query returns the history of VPN links for various CoS (Class of Service). It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **source**—The source of the VPN link for a CoS whose history needs to be provided. This can be the router name or the IP address of the router.
- **destination**—The destination of the VPN link for a CoS whose history needs to be provided. This can be the router name or the IP address of the router.
- **cos**—The Class of Service (CoS) for the traffic.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time

- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- destination: string
- cos: string
- array of following VPN links CoS history statistic structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[2];
my $topoDB = $ARGV[1];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $SRlink = $ARGV[5];
my $Dstlink = $ARGV[6];
my $COS = $ARGV[7];
my $StatsType = $ARGV[8];
my $TimeRange = $ARGV[9];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
```

```
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);


push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_cos_vpn_history
',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($SRlink),
                            RPC::XML::RPC_STRING($Dstlink),
                            RPC::XML::RPC_STRING($COS),
                            RPC::XML::RPC_STRING($StatsType),
                            RPC::XML::RPC_STRING($TimeRange))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T21:53:30',
  'totalEntries' => '1',
  'result' => {
```

```
    'history_links_cos_vpn' => {
      'source' => 'SF-CORE-ROUTER2',
      'end_time' => '20081015T22:00:00',
      'cos' => 'Exp1',
      'destination' => 'SF-PE2-ROUTER8.01',
      'statistics' => [
        {
          'average' => '2452665',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_links_history

RPC Call: TrafficAnalyzer.api_traffic_links_history {password} {database} {start time} {end time} {source} {destination} {statistics} {report time range}

This query returns the history of the aggregate links for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **source**—The source of the aggregate link whose history needs to be provided. This can be the router name or the IP address of the router.
- **destination**—The destination of the aggregate link whose history needs to be provided. This can be the router name or the IP address of the router.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- destination: string

- array of following aggregate links history statistics structure:
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $startTime = str2time("20081014T10:00:00");
my $endTime = str2time("20081015T15:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("CORE-ROUTER13"),
                            RPC::XML::RPC_STRING("CORE-ROUTER13.03"),
                            RPC::XML::RPC_STRING("Average"),
```

```
                                RPC::XML::RPC_STRING("daily"))
                                );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T19:12:42',
  'totalEntries' => '1',
  'result' => {
    'history_links' => {
      'source' => 'CORE-ROUTER13',
      'end_time' => '20081015T22:00:00',
      'destination' => 'CORE-ROUTER13.03',
      'statistics' => [
        {
          'average' => '1953584',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_links_ipv4

RPC Call: TrafficAnalyzer.api_traffic_links_ipv4 {password} {database name} {time} {report time range}

This query returns the list of traffic statistics for IPv4 links for a particular time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output:

- vinfo: version struct

- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedentries: int
- total entries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following link statistics structure :
  - source: string
  - destination: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";
```

```
    my $t1 = str2time("20081020T12:30:00");

    push (@reqs,
    RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_ipv4',
                                RPC::XML::RPC_STRING($password),
                                RPC::XML::RPC_STRING($database),
                           RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                                RPC::XML::RPC_STRING("hourly"))
                                );

    foreach (@reqs) {
            my $res = $client->send_request($_);
            if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
            my $value1 = $res->value;

            print Dumper($value1);
}
```

Sample Output

```
    {
      'vinfo' => {
        'software_version' => '8.0.30-R Traffic Explorer',
        'appliance_version' => '8.0.30-R'
      },
      'numReturnedEntries' => '2',
      'network_name' => 'PDlab',
      'report_time' => '20081020T23:53:18',
      'totalEntries' => '2',
      'result' => {
        'report_result' => [
          {
            'source' => 'SF-CORE-RTR1',
            'destination' => 'CORE-ROUTER13.03',
            'avg' => '12183231',
            'min' => '9074376',
            'max' => '13817500',
            'ninetyfifthpctile' => '13817500'
          },
          {
            'source' => 'CORE-ROUTER13.03',
            'destination' => 'CORE-ROUTER13',
```

```
          'avg' => '12183231',
          'min' => '9074376',
          'max' => '13817500',
          'ninetyfifthpctile' => '13817500'
        }
      ],
      'report_start_time' => '20081020T18:00:00',
      'report_end_time' => '20081020T18:59:59'
    }
  }
```

api_traffic_links_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_links_ipv4_history {password} {database} {start time) {end time} {source} {destination} {statistics} {report time range}

This query returns the history of IPv4 links for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **source**—The source of the IPv4 link whose history needs to be provided. This can be the router name or the IP address of the router.
- **destination**—The destination of the IPv4 link whose history needs to be provided. This can be the router name or the IP address of the router.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- destination: string
- array of following IPv4 links history statistics structure:
  - time: ISO 8601 UTC time

— statistics: double (bps)

— start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $startTime = str2time("20081014T10:00:00");
my $endTime = str2time("20081015T15:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_ipv4_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("CORE-ROUTER13"),
                            RPC::XML::RPC_STRING("CORE-ROUTER13.03"),
                            RPC::XML::RPC_STRING("Average"),
                            RPC::XML::RPC_STRING("daily"))
                            );
```

```
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T18:16:21',
  'totalEntries' => '1',
  'result' => {
    'history_ipv4_links' => {
      'source' => 'CORE-ROUTER13',
      'end_time' => '20081015T22:00:00',
      'destination' => 'CORE-ROUTER13.03',
      'statistics' => [
        {
          'average' => '743',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_links_vpn

RPC Call: TrafficAnalyzer.api_traffic_links_vpn {password} {database name} {time} {report time range}

This query returns the traffic statistics for VPN links for a particular requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time

- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of link statistics structure containing the following :
  - source: string
  - destination: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2002/RPC2";

my $t1 = str2time("20081020T12:30:00");
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_vpn',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),
                       RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081020T23:57:57',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'source' => 'DC-CORE1-ROUTER3',
        'destination' => 'ROUTER11.05',
        'avg' => '30755199',
        'min' => '26150669',
        'max' => '34390072',
        'ninetyfifthpctile' => '34390072'
      },
      {
        'source' => 'ROUTER11.05',
        'destination' => 'ROUTER11',
        'avg' => '30755199',
        'min' => '26150669',
```

```
          'max' => '34390072',
          'ninetyfifthpctile' => '34390072'
        }
    ],
    'report_start_time' => '20081020T18:00:00',
    'report_end_time' => '20081020T18:59:59'
  }
}
```

api_traffic_links_vpn_history

RPC Call: TrafficAnalyzer.api_traffic_links_vpn_history {password} {database name} {start time} {end time} {source} {destination} {statistics} {report time range}

This query returns the history of the VPN links for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.

- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **source**—The source of the VPN link whose history needs to be provided. The source can be the router or the IP address of the router.
- **destination**—The destination of the VPN link whose history needs to be provided. The source can be the router or the IP address of the router.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- source: string
- end_time: ISO 8601 UTC time
- destination: string
- array of following VPN links history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $startTime = str2time("20081014T10:00:00");
my $endTime = str2time("20081015T15:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_links_vpn_history',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($startTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($endTime)),
                            RPC::XML::RPC_STRING("CORE-ROUTER13"),
                            RPC::XML::RPC_STRING("CORE-ROUTER13.03"),
                            RPC::XML::RPC_STRING("Average"),
                            RPC::XML::RPC_STRING("daily"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
```

```
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDlab',
  'report_time' => '20081023T19:08:51',
  'totalEntries' => '1',
  'result' => {
    'history_vpn_links' => {
      'source' => 'CORE-ROUTER13',
      'end_time' => '20081015T22:00:00',
      'destination' => 'CORE-ROUTER13.03',
      'statistics' => [
        {
          'average' => '1952841',
          'time' => '20081015T07:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_list_flows

RPC Call: TrafficAnalyzer.api_traffic_list_flows {password} {database name} {time} {exporting index} {topn}

This query returns a list of prefix-aggregated flows along with other routing attributes for the specified 5 minute interval. It also accepts exporting interface index and topn count as optional filter parameters.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time. Report time range is 5 minute interval. If user specified time is not on 5 minute interval, then data for next 5 minute interval will be returned. If traffic data is not available at next 5 minute interval, error code 352 (`specified time is later than the latest available traffic data`) will be returned.
- **exporting index**—(Optional) List of exporter ip and interface index in comma separated format such as `10.120.1.1:2, 10.120.1.2:5, 10.120.1.3:1`. Empty string indicates no filtering is needed on exporting interfaces. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.
- **topn**—Number of top entries that should be returned in output.

Structure of Output

- vinfo:versionstruct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- total entries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- customer report result: array of the following structures. We return "fraction" (double) with each route aggregate which indicates fraction of flow associated with it.
- source: (prefix struct)
- destination: (prefix struct)

- exporter
  - address (ip address struct)
  - index (int)
  - fraction (double)
- traffic_group
  - name (string)
  - fraction (double)
- cos
  - cos_grp_name (string)
  - fraction (double)
- frr
  - name (string)
  - label (int)
  - frr_head (ip address struct)
  - fraction (double)
- tunnel
  - name (string)
  - label (int)
  - tunnel_head (ip address struct)
  - fraction (double)
  - egress_pe
  - egress_pe_label (int)
  - egress_pe (prefix)
  - fraction (double)
  - egress_vrf
  - name (string)
  - label (int)
  - fraction (double)

- ingress_vrf
- ingress_vrf (string)
- fraction (double)
- ingress_pe
- ingress_pe (prefix)
- fraction (double)
- vpn_customer
- vpn_customer (string)
- fraction (double)

- bgp
- source_as
  - number (int)
  - name (string)
  - fraction (double)
- neighbor_as
  - number (int)
  - name (string)
  - fraction (double)
- destination_as
  - number (int)
  - name (string)
  - fraction (double)
- transit_as
  - number (int)
  - name (string)
  - fraction (double)
- exit_router
  - exit_router_ip (ip address struct)

- fraction (double)

- bgp_nexthop
  - bgp_nexthop (ip address struct)
  - fraction (double)
- outgoing_interface_index
  - index (int)
  - fraction (double)
- flow_recorder
  - flow_recorder_ip (ip address struct)
  - fraction (double)
- path
  - num_hops (int)
  - path_source (router struct)
  - path_destination (prefix struct)
  - fraction (double)
  - path_hops (below 4 tags of tunnel information will be present only if it is a tunnel hop. If hop is of frr type, then below tag names are frr_name, source, destination, frr_hops)

    - tunnel_name (string)

    - source (ip address struct)

    - destination (ip address struct)

    - fraction (double)
  - tunnel_hops

    - hop_src (router struct)

    - hop_dst (router struct)

    - vpn (only present for vpn flow)

      - name (string)

      - fraction (double)

    interfaces (present only if it is tunnel hop)

- sif (ip address struct)
- dif (ip address struct)
- fraction (double)

- traffic_5min_avg (5 minute traffic average in bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $Filter1 = $ARGV[4];
my $Filter2 = $ARGV[5];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper;

$Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:12002/RPC2";

my $StartTime = str2time($StartTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_list_flows',
                                    RPC::XML::RPC_STRING($PassWord),
                                    RPC::XML::RPC_STRING($topoDB),
```

```
RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),
                                    RPC::XML::RPC_STRING($Filter1),
                                    RPC::XML::RPC_INT($Filter2)) );

foreach (@reqs) {
    my $res = $client->send_request($_);
    if ($res->is_fault) { print("---XMLRPC FAULT ---"); }
    my $value1 = $res->value;
    print Dumper($value1);
}

}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => '9.3.16'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'PDI',
  'network_time' => '20111022T00:00:00',
  'report_time' => '20111022T09:01:51',
  'totalEntries' => '1',
  'result' => {
    'report_result' => [
      {
        'source' => {
          'masklen' => '32',
          'ip_addr' => {
            'ip4_addr' => '10.120.1.9'
          }
        },
        'destination' => {
          'masklen' => '32',
          'ip_addr' => {
            'ip4_addr' => '10.120.1.8'
          }
        },
        'traffic_group' => {
```

```
  'name' => 'T-LDP-CTL-ISIS-Core-to-Router',
  'fraction' => '0'
},
'bgp' => {
  'transit_as' => [],
  'exit_router' => {
    'exit_router_ip' => {
      'ip4_addr' => '10.120.1.8'
    },
    'fraction' => '0'
  }
},
'flow_recorder' => {
  'flow_recorder_ip' => {
    'ip4_addr' => '192.168.3.53'
  },
  'fraction' => '1'
},
'path' => {
  'path_dst' => {
    'masklen' => '32',
    'ip_addr' => {
      'ip4_addr' => '10.120.1.8'
    }
  },
  'path_hops' => [
    {
      'tunnel_name' => 'Tunnel9866',
      'source' => {
        'ip4_addr' => '10.120.1.9'
      },
      'destination' => {
        'ip4_addr' => '10.120.1.8'
      },
      'fraction' => '1',
      'tunnel_hops' => [
        {
          'hop_src' => {
            'name' => 'SJ-PE-2811-R9',
            'model' => 'Cisco',
            'softwareVersion' => 'IOS',
            'serialNumber' => 'FTX1104A04J',
```

```
      'mpname' => 'SJ-PE-2811-R9',
      'mpID' => '0x0101200010090000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.9'
      }
    },
    'hop_dst' => {
      'mpname' => 'SJ-PE-2811-R9.03',
      'mpID' => '0x0A400C0000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.12.0'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.12.0'
        }
      }
    },
    'interfaces' => {
      'dif' => 'Unnumbered',
      'sif' => {
        'ip4_addr' => '10.64.12.9'
      },
      'fraction' => '1'
    },
    'fraction' => '1'
  },
  {
    'hop_src' => {
      'mpname' => 'SJ-PE-2811-R9.03',
      'mpID' => '0x0A400C0000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.12.0'
      },
```

```
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.12.0'
        }
      }
    },
    'hop_dst' => {
      'name' => 'SJ-P-7204-R4',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => '26808117',
      'mpname' => 'SJ-P-7204-R4',
      'mpID' => '0x0550550550550000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.4'
      }
    },
    'interfaces' => {
      'dif' => {
        'ip4_addr' => '10.64.12.4'
      },
      'sif' => 'Unnumbered',
      'fraction' => '1'
    },
    'fraction' => '1'
  },
  {
    'hop_src' => {
      'name' => 'SJ-P-7204-R4',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => '26808117',
      'mpname' => 'SJ-P-7204-R4',
      'mpID' => '0x0550550550550000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.4'
      }
    },
    'hop_dst' => {
```

```
      'mpname' => 'SJ-P-7204-R4.05',
      'mpID' => '0x0A40070000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.7.0'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.7.0'
        }
      }
    },
    'interfaces' => {
      'dif' => 'Unnumbered',
      'sif' => {
        'ip4_addr' => '10.64.7.4'
      },
      'fraction' => '1'
    },
    'fraction' => '1'
  },
  {
    'hop_src' => {
      'mpname' => 'SJ-P-7204-R4.05',
      'mpID' => '0x0A40070000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.7.0'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.7.0'
        }
      }
    },
    'hop_dst' => {
```

```
      'name' => 'SJ-P-7204-R5',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => '30461365',
      'mpname' => 'SJ-P-7204-R5',
      'mpID' => '0x0101200010050000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.5'
      }
    },
    'interfaces' => {
      'dif' => {
        'ip4_addr' => '10.64.7.5'
      },
      'sif' => 'Unnumbered',
      'fraction' => '1'
    },
    'fraction' => '1'
  },
  {
    'hop_src' => {
      'name' => 'SJ-P-7204-R5',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => '30461365',
      'mpname' => 'SJ-P-7204-R5',
      'mpID' => '0x0101200010050000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.5'
      }
    },
    'hop_dst' => {
      'mpname' => 'SJ-P-7204-R2.EF',
      'mpID' => '0x0A40030000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.3.0'
      },
```

```
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.3.0'
        }
      }
    },
    'interfaces' => {
      'dif' => 'Unnumbered',
      'sif' => {
        'ip4_addr' => '10.64.3.5'
      },
      'fraction' => '0'
    },
    'fraction' => '0'
  },
  {
    'hop_src' => {
      'mpname' => 'SJ-P-7204-R2.EF',
      'mpID' => '0x0A40030000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.3.0'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.3.0'
        }
      }
    },
    'hop_dst' => {
      'name' => 'SJ-P-7204-R2',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => '29495818',
      'mpname' => 'SJ-P-7204-R2',
      'mpID' => '0x0101200010020000',
      'type' => 'Device',
      'ipaddr' => {
```

```
      'ip4_addr' => '10.120.1.2'
    }
  },
  'interfaces' => {
    'dif' => {
      'ip4_addr' => '10.64.3.2'
    },
    'sif' => 'Unnumbered',
    'fraction' => '0'
  },
  'fraction' => '0'
},
{
  'hop_src' => {
    'name' => 'SJ-P-7204-R2',
    'model' => 'Cisco',
    'softwareVersion' => 'IOS',
    'serialNumber' => '29495818',
    'mpname' => 'SJ-P-7204-R2',
    'mpID' => '0x0101200010020000',
    'type' => 'Device',
    'ipaddr' => {
      'ip4_addr' => '10.120.1.2'
    }
  },
  'hop_dst' => {
    'mpname' => 'SJ-PE-2811-R8.02',
    'mpID' => '0x0A400B0000000104',
    'model' => '',
    'type' => 'LAN Pseudo-Node',
    'softwareVersion' => '',
    'ipaddr' => {
      'ip4_addr' => '10.64.11.0'
    },
    'prefix' => {
      'masklen' => '24',
      'ip_addr' => {
        'ip4_addr' => '10.64.11.0'
      }
    }
  },
  'interfaces' => {
```

```
      'dif' => 'Unnumbered',
      'sif' => {
        'ip4_addr' => '10.64.11.2'
      },
      'fraction' => '0'
    },
    'fraction' => '0'
  },
  {
    'hop_src' => {
      'mpname' => 'SJ-PE-2811-R8.02',
      'mpID' => '0x0A400B0000000104',
      'model' => '',
      'type' => 'LAN Pseudo-Node',
      'softwareVersion' => '',
      'ipaddr' => {
        'ip4_addr' => '10.64.11.0'
      },
      'prefix' => {
        'masklen' => '24',
        'ip_addr' => {
          'ip4_addr' => '10.64.11.0'
        }
      }
    },
    'hop_dst' => {
      'name' => 'SJ-PE-2811-R8',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => 'FTX1104A1LH',
      'mpname' => 'SJ-PE-2811-R8',
      'mpID' => '0x0101200010080000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.8'
      }
    },
    'interfaces' => {
      'dif' => {
        'ip4_addr' => '10.64.11.8'
      },
      'sif' => 'Unnumbered',
```

```
              'fraction' => '0'
            },
            'fraction' => '0'
          }
        ]
      }
    ],
    'path_src' => {
      'name' => 'SJ-PE-2811-R9',
      'model' => 'Cisco',
      'softwareVersion' => 'IOS',
      'serialNumber' => 'FTX1104A04J',
      'mpname' => 'SJ-PE-2811-R9',
      'mpID' => '0x0101200010090000',
      'type' => 'Device',
      'ipaddr' => {
        'ip4_addr' => '10.120.1.9'
      }
    },
    'num_hops' => '9'
  },
  'traffic_5min_avg' => '16',
  'cos' => {
    'cos_grp_name' => 'SAP',
    'fraction' => '0'
  },
  'tunnel' => {
    'name' => 'Tunnel9866',
    'label' => '1766',
    'tunnel_head' => {
      'ip4_addr' => '10.120.1.9'
    },
    'fraction' => '0'
  },
  'exporter' => {
    'index' => '1',
    'address' => {
      'ip4_addr' => '10.120.1.4'
    },
    'fraction' => '1'
  }
},
```

```
    'report_start_time' => '20111021T23:55:00',
    'report_end_time' => '20111022T00:00:00'
  }
}
```

api_traffic_neighbor_as_ipv4

RPC Call: TrafficAnalyzer.api_traffic_neighbor_as_ipv4 {password} {database name} {time} {report time range}

This query returns IPv4 BGP neighbor AS statistics from traffic reports for the requested time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following neighbor AS statistics structure :
  - neighbor_as: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);
```

```
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_neighbor_as_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '3',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:56',
  'totalEntries' => '3',
```

```
  'result' => {
    'report_result' => [
      {
        'avg' => '2472940',
        'min' => '1750429',
        'max' => '2742540',
        'ninetyfifthpctile' => '2742540',
        'neighbor_as' => 'OSPF (65471)'
      },
      {
        'avg' => '338739',
        'min' => '238202',
        'max' => '357900',
        'ninetyfifthpctile' => '357900',
        'neighbor_as' => 'ISIS (65473)'
      },
      {
        'avg' => '147',
        'min' => '0',
        'max' => '215',
        'ninetyfifthpctile' => '215',
        'neighbor_as' => 'private (65470)'
      }
    ],
    'report_start_time' => '20081020T11:00:00',
    'report_end_time' => '20081020T11:59:59'
  }
}
```

api_traffic_neighbor_as_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_neighbor_as_ipv4_history {password} {database} {start time} {end time} {as num} {statistics} {report time range}

This query returns the history of IPV4 Neighbor AS for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35.`
- **as num—**The AS number whose history needs to be provided.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string

- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of following IPv4 neighbor AS history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - neighbor_as: int

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $AS = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";
```

```
my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_neighbor_as_ipv4_hist
ory',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_INT($AS),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:22:08',
  'totalEntries' => '2',
  'result' => {
    'history_neighbor_as' => {
      'end_time' => '20081014T18:00:00',
      'statistics' => [
        {
          'time' => '20081014T17:00:00',
          'avg' => '3275795'
```

```
          },
          {
            'time' => '20081014T18:00:00',
            'avg' => '3405443'
          }
        ],
        'start_time' => '20081014T17:00:00',
        'neighbor_as' => '65473'
      }
    }
}
```

api_traffic_source_as_ipv4

RPC Call: TrafficAnalyzer.api_traffic_source_as_ipv4 {password} {database name} {time} {report time range}

This query returns the IPv4 BGP source AS statistics from traffic reports for the requested time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following source AS statistics structure :
  - source_as: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
```

```
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_source_as_ipv4',
                           RPC::XML::RPC_STRING($password),
                           RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                           RPC::XML::RPC_STRING("hourly"))
                           );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
```

```
'vinfo' => {
  'software_version' => '8.0.30-R Traffic Explorer',
  'appliance_version' => '8.0.30-R'
},
'numReturnedEntries' => '7',
'network_name' => 'PDlab',
'report_time' => '20081021T03:07:56',
'totalEntries' => '7',
'result' => {
  'report_result' => [
    {
      'source_as' => 'INTERNAL',
      'avg' => '41409820',
      'min' => '38881823',
      'max' => '45807521',
      'ninetyfifthpctile' => '45807521'
    },
    {
      'source_as' => 'private (65300)',
      'avg' => '61956',
      'min' => '57486',
      'max' => '64273',
      'ninetyfifthpctile' => '64273'
    },
    {
      'source_as' => 'private (64520)',
      'avg' => '59437',
      'min' => '45133',
      'max' => '70551',
      'ninetyfifthpctile' => '70551'
    },
    {
      'source_as' => 'private (64900)',
      'avg' => '55128',
      'min' => '52678',
      'max' => '56436',
      'ninetyfifthpctile' => '56436'
    },
    {
      'source_as' => 'private (64550)',
      'avg' => '37323',
      'min' => '23305',
```

```
          'max' => '43267',
          'ninetyfifthpctile' => '43267'
        },
        {
          'source_as' => 'private (64590)',
          'avg' => '29511',
          'min' => '27172',
          'max' => '30598',
          'ninetyfifthpctile' => '30598'
        },
        {
          'source_as' => 'OSPF (65471)',
          'avg' => '2008',
          'min' => '1147',
          'max' => '2482',
          'ninetyfifthpctile' => '2482'
        }
      ],
      'report_start_time' => '20081020T11:00:00',
      'report_end_time' => '20081020T11:59:59'
    }
}
```

api_traffic_source_as_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_source_as_ipv4_history {password} {database} {start time} {end time} {as num} {statistics} {report time range}

This query returns the history of IPv4 source AS for the requested time period. It will retrieve the history of the average, minimum, maximum, or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.

- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **as num—**The AS number whose history will be retrieved.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- source_as: integer
- array of following IPv4 source as history statistics structure
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
```

```
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $AS = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_source_as_ipv4_histor
y',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_INT($AS),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:24:58',
  'totalEntries' => '2',
  'result' => {
    'history_source_as' => {
      'end_time' => '20081014T18:00:00',
      'source_as' => '65471',
      'statistics' => [
        {
          'percentile' => '3104',
          'time' => '20081014T17:00:00'
        },
        {
          'percentile' => '3100',
          'time' => '20081014T18:00:00'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_src_nbr_matrix

RPC Call: TrafficAnalyzer.api_traffic_src_nbr_matrix {password} {database name} {time} {source AS} {neighbor AS} {exporting index}

This query returns all combinations of Exporting router, Exporting interface index, source AS, neighbor AS for the specified 5 minute interval. It also accepts list of source AS, neighbor AS and exporting interface index as optional filter parameters.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time. Report time range is 5 minute interval. If user specified time is not on 5 minute interval, then data for next 5 minute interval will be returned. If traffic data is not available at next 5 minute interval, error code 352 (`specified time is later than the latest available traffic data`) will be returned.
- **source AS**—(Optional) List of source AS numbers in comma separated format, such as `11111, 22222, 33333, 44444`. Empty string indicates no filtering needed on source AS. If this parameter is present, output will be generated only for specified source AS numbers. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.
- **neighbor AS**—(Optional) List of neighbor AS numbers in comma separated format such as `11111, 22222, 33333, 44444`. Empty string indicates no filtering needed on neighbor AS. If this parameter is present, output will be generated only for specified neighbor AS numbers. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.
- **exporting index**—(Optional) List of exporter ip and interface index in comma separated format such as `10.120.1.1:2, 10.120.1.2:5, 10.120.1.3:1`. Empty string indicates no filtering is needed on exporting interfaces. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.

Structure of Output

- vinfo:versionstruct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- total entries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time

- customer report result: array of the following structures:
  - source: string
  - destination: string
  - interface
    - index (int)
    - name (string)
    - address (prefix struct)
    - capacity (bps)
    - description (string)
  - source_as
    - number (int)
    - name (string)
  - neighbor_as
    - number (int)
    - name (string)
  - avg (bps) - 5 minute traffic average

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $Filter1 = $ARGV[4];
my $Filter2 = $ARGV[5];
my $Filter3 = $ARGV[6];

use strict;
use RPC::XML::Client;
```

```
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper;

$Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:12002/RPC2";

my $StartTime = str2time($StartTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_src_nbr_matrix',
                                    RPC::XML::RPC_STRING($PassWord),
                                    RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),
                                    RPC::XML::RPC_STRING($Filter1),
                                    RPC::XML::RPC_STRING($Filter2),
                                    RPC::XML::RPC_STRING($Filter3)) );

foreach (@reqs) {
    my $res = $client->send_request($_);
    if ($res->is_fault) { print("---XMLRPC FAULT ---"); }
    my $value1 = $res->value;
    print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => '9.4.13'
  },
  'numReturnedEntries' => '1',
  'network_name' => 'hermes',
  'network_time' => '20111020T17:00:00',
```

```
  'report_time' => '20111021T13:29:37',
  'totalEntries' => '1',
  'result' => {
    'report_result' => [
      {
        'source' => 'SJ-P-M5-R12.04',
        'source_as' => {
          'number' => '0',
          'name' => 'No AS'
        },
        'destination' => 'SJ-P-7204-R5',
        'avg' => '1420000',
        'interface' => {
          'index' => '2',
          'name' => 'Fa1/1',
          'address' => {
            'masklen' => '24',
            'ip_addr' => {
              'ip4_addr' => '10.64.20.5'
            }
          },
          'capacity' => '100000000',
          'description' => 'R5-R12-Fa1/1-10.64.20.5'
        },
        'neighbor_as' => {
          'number' => '65477',
          'name' => ' -private use as-'
        }
      },
    'report_start_time' => '20111020T16:55:00',
    'report_end_time' => '20111020T17:00:00'
  }
}
```

api_traffic_src_dst_matrix

RPC Call: TrafficAnalyzer.api_traffic_src_dst_matrix {password} {database name} {time} {source AS} {destination AS} {exporting index}

This query returns all combinations of Exporting router, Exporting interface index, source AS, destination AS for the specified 5 minute interval. It also accepts list of source AS, destination AS and exporting interface index as optional filter parameters.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time**—A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time. Report time range is 5 minute interval. If user specified time is not on 5 minute interval, then data for next 5 minute interval will be returned. If traffic data is not available at next 5 minute interval, error code 352 (`specified time is later than the latest available traffic data`) will be returned.
- **source AS**—(Optional) List of source AS numbers in comma separated format, such as `11111, 22222, 33333, 44444`. Empty string indicates no filtering needed on source AS. If this parameter is present, output will be generated only for specified source AS numbers. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.
- **destination AS**—(Optional) List of destination AS numbers in comma separated format such as `11111, 22222, 33333, 44444`. Empty string indicates no filtering needed on neighbor AS. If this parameter is present, output will be generated only for specified neighbor AS numbers. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.
- **exporting index**—(Optional) List of exporter ip and interface index in comma separated format such as `10.120.1.1:2, 10.120.1.2:5, 10.120.1.3:1`. Empty string indicates no filtering is needed on exporting interfaces. If there is formatting error in input string, then error code 353 (`Invalid filter argument format`) is returned.

Structure of Output

- vinfo:versionstruct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- total entries: int

- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- customer report result: array of the following structures:
  - source: string
  - destination: string
  - interface
    - index (int)
    - name (string)
    - address (prefix struct)
    - capacity (bps)
    - description (string)
  - source_as
    - number (int)
    - name (string)
  - destination_as
    - number (int)
    - name (string)
  - avg (bps) - 5 minute traffic average

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $Filter1 = $ARGV[4];
my $Filter2 = $ARGV[5];
my $Filter3 = $ARGV[6];
```

```
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper;

$Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:12002/RPC2";

my $StartTime = str2time($StartTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_src_dst_matrix',
                                    RPC::XML::RPC_STRING($PassWord),
                                    RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),
                                    RPC::XML::RPC_STRING($Filter1),
                                    RPC::XML::RPC_STRING($Filter2),
                                    RPC::XML::RPC_STRING($Filter3)) );

foreach (@reqs) {
    my $res = $client->send_request($_);
    if ($res->is_fault) { print("---XMLRPC FAULT ---"); }
    my $value1 = $res->value;
    print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => '9.4.13'
  },
```

```
  'numReturnedEntries' => '1',
  'network_name' => 'hermes',
  'network_time' => '20111020T17:00:00',
  'report_time' => '20111021T13:36:28',
  'totalEntries' => '1',
  'result' => {
    'report_result' => [
      {
        'source' => 'SJ-P-M5-R12.04',
        'source_as' => {
          'number' => '0',
          'name' => 'No AS'
        },
        'destination' => 'SJ-P-7204-R5',
        'avg' => '1420000',
        'interface' => {
          'index' => '2',
          'name' => 'Fa1/1',
          'address' => {
            'masklen' => '24',
            'ip_addr' => {
              'ip4_addr' => '10.64.20.5'
            }
          },
          'capacity' => '100000000',
          'description' => 'R5-R12-Fa1/1-10.64.20.5'
        },
        'destination_as' => {
          'number' => '65477',
          'name' => ' -private use as-'
        }
      },
    'report_start_time' => '20111020T16:55:00',
    'report_end_time' => '20111020T17:00:00'
  }
}
```

api_traffic_traffic_groups_ipv4

RPC Call: TrafficAnalyzer.api_traffic_traffic_groups_ipv4 {password} {database name} {time} {report time range}

This query returns IPv4 traffic group statistics from traffic reports for the requested time frame. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following traffic group statistics structure :
  - traffic_group: string
  - avg: double (bps)
  - min: double (bps)

— max: double (bps)

— ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_traffic_groups_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
```

```
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:57',
  'totalEntries' => '2',
  'result' => {
    'report_result' => [
      {
        'traffic_group' => 'Other',
        'avg' => '41654783',
        'min' => '39111629',
        'max' => '46042367',
        'ninetyfifthpctile' => '46042367'
      },
      {
        'traffic_group' => 'ldp',
        'avg' => '403',
        'min' => '238',
        'max' => '566',
        'ninetyfifthpctile' => '566'
      }
    ],
    'report_start_time' => '20081020T11:00:00',
    'report_end_time' => '20081020T11:59:59'
  }
}
```

api_traffic_traffic_groups_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_traffic_groups_ipv4_history {password} {database} {start time} {end time} {traffic grp} {statistics} {report time range}

This query returns the history of IPv4 traffic groups for the requested time period. It will retrieve the history of the minimum, maximum, average or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35.
- **traffic grp**—The IPv4 traffic group whose history will be returned.
- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.

- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- traffic_group: string
- array of following IPv4 traffic groups history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $TrafficGroupName = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
my $TimeRange = $ARGV[7];

use strict;
```

```
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_traffic_groups_ipv4_h
istory',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_STRING($TrafficGroupName),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '2',
```

```
  'network_name' => 'PDlab',
  'report_time' => '20081023T20:38:28',
  'totalEntries' => '2',
  'result' => {
    'history_traffic_groups_ipv4' => {
      'end_time' => '20081014T18:00:00',
      'traffic_group' => 'ldp',
      'statistics' => [
        {
          'time' => '20081014T17:00:00',
          'avg' => '363'
        },
        {
          'time' => '20081014T18:00:00',
          'avg' => '366'
        }
      ],
      'start_time' => '20081014T17:00:00'
    }
  }
}
```

api_traffic_transit_as_ipv4

RPC Call: TrafficAnalyzer.api_traffic_transit_as_ipv4 {password} {database name} {time} {report time range}

This query returns the IPv4 BGP transit AS statistics from traffic reports for the requested time. The statistics include the average, minimum, maximum, and 95 percentile in bits per second (bps).

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results will be calculated based on the network state at the specified time.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- numReturnedEntries: int
- totalEntries: int
- report_start_time: ISO 8601 UTC time
- report_end_time: ISO 8601 UTC time
- array of the following transit AS statistics structure :
  - transit_as: string
  - avg: double (bps)
  - min: double (bps)
  - max: double (bps)
  - ninetyfifthpctile: double (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $qsip = $ARGV[0];
my $database = $ARGV[1];
my $filter = "any";

$filter = $ARGV[2] if ($#ARGV >= 2);

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $ReportTime = str2time("20081020T05:00:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_transit_as_ipv4',
                            RPC::XML::RPC_STRING($password),
                            RPC::XML::RPC_STRING($database),

RPC::XML::datetime_iso8601->new(time2iso8601($ReportTime)),
                            RPC::XML::RPC_STRING("hourly"))
                            );

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
```

```
}
```

Sample Output

```
'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'numReturnedEntries' => '3',
  'network_name' => 'PDlab',
  'report_time' => '20081021T03:07:56',
  'totalEntries' => '3',
  'result' => {
    'report_result' => [
      {
        'avg' => '2030182',
        'min' => '1302872',
        'transit_as' => 'OSPF (65471)',
        'max' => '2290637',
        'ninetyfifthpctile' => '2290637'
      },
      {
        'avg' => '874995',
        'min' => '668529',
        'transit_as' => 'private (64520)',
        'max' => '1027233',
        'ninetyfifthpctile' => '1027233'
      },
      {
        'avg' => '338739',
        'min' => '238202',
        'transit_as' => 'ISIS (65473)',
        'max' => '357900',
        'ninetyfifthpctile' => '357900'
      }
    ],
    'report_start_time' => '20081020T11:00:00',
    'report_end_time' => '20081020T11:59:59'
  }
}
```

api_traffic_transit_as_ipv4_history

RPC Call: TrafficAnalyzer.api_traffic_transit_as_ipv4_history {password} {database name} {start time} {end time} {as num} {statistics} {report time range}

This query returns the history of IPv4 transit AS for the requested time period. It will retrieve the history of the average, minimum, maximum or percentile statistics that has been requested.

Input Parameters

- **password**—The password configured for queries.
- **database name—**A name from the database hierarchy, which may be an administrative domain such as CorpNet, which includes the subtree below it, or a complete database name, such as CorpNet.EIGRP/AS100.
- **start time**—The starting time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **end time**—The ending time that history data needs to return. It is specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`.
- **as num—**The AS number whose history needs to be provided.

- **statistics**—The statistic to be reported, which may be any one of the following: average, minimum, maximum, or percentile; percentile means 95th percentile.
- **report time range**—The interval over which the reported statistic was calculated for each data point. The time range can be any one of the following: any, hourly, daily, weekly, or monthly; any means 5-minute intervals.

Structure of Output

- vinfo: version struct
- numReturnedEntries: int
- network_name: string
- report_time: ISO 8601 UTC time
- totalEntries: int
- name_of_history: string
- end_time: ISO 8601 UTC time
- array of the following IPv4 transit AS history statistics structure :
  - time: ISO 8601 UTC time
  - statistics: double (bps)
  - start_time: ISO 8601 UTC time
  - transit_as: integer

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    exit(0);
}

my $IpAddress = $ARGV[0];
my $PassWord = $ARGV[1];
my $topoDB = $ARGV[2];
my $StartTimeOfReport = $ARGV[3];
my $EndTimeOfReport = $ARGV[4];
my $AS = $ARGV[5];
my $TypeOfStatistics = $ARGV[6];
```

```
my $TimeRange = $ARGV[7];

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;

my $client;
my $req;
my @reqs;
$client = new RPC::XML::Client "http://$IpAddress:2000/RPC2";

my $StartTime = str2time($StartTimeOfReport);
my $EndTime = str2time($EndTimeOfReport);

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_transit_as_ipv4_histo
ry',
                            RPC::XML::RPC_STRING($PassWord),
                            RPC::XML::RPC_STRING($topoDB),

RPC::XML::datetime_iso8601->new(time2iso8601($StartTime)),

RPC::XML::datetime_iso8601->new(time2iso8601($EndTime)),
                            RPC::XML::RPC_INT($AS),
                            RPC::XML::RPC_STRING($TypeOfStatistics),
                            RPC::XML::RPC_STRING($TimeRange))
                            );
foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;

        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
```

```
  'appliance_version' => '8.0.30-R'
},
'numReturnedEntries' => '13',
'network_name' => 'PDlab',
'report_time' => '20081023T20:21:28',
'totalEntries' => '13',
'result' => {
  'history_transit_as' => {
    'end_time' => '20081014T18:00:00',
    'statistics' => [
      {
        'time' => '20081014T17:00:00',
        'avg' => '3728578'
      },
      {
        'time' => '20081014T17:05:00',
        'avg' => '3300781'
      },
      {
        'time' => '20081014T17:10:00',
        'avg' => '2916058'
      },
      {
        'time' => '20081014T17:15:00',
        'avg' => '3298302'
      },
      {
        'time' => '20081014T17:20:00',
        'avg' => '3623895'
      },
      {
        'time' => '20081014T17:25:00',
        'avg' => '3737390'
      },
      {
        'time' => '20081014T17:30:00',
        'avg' => '3110190'
      },
      {
        'time' => '20081014T17:35:00',
        'avg' => '3332316'
      },
```

```
          {
            'time' => '20081014T17:40:00',
            'avg' => '3493996'
          },
          {
            'time' => '20081014T17:45:00',
            'avg' => '3250862'
          },
          {
            'time' => '20081014T17:50:00',
            'avg' => '3569558'
          },
          {
            'time' => '20081014T17:55:00',
            'avg' => '3736687'
          },
          {
            'time' => '20081014T18:00:00',
            'avg' => '3495278'
          }
        ],
        'start_time' => '20081014T17:00:00',
        'transit_as' => '65473'
      }
    }
  }
```

7 Configuration Queries

This chapter describes the api_manage_config query. This query will eventually allow exporting and importing the complete system configuration or any portion thereof.

Overview

The appliance configuration is formatted in XML as defined by the configuration schema, which includes a specification of the operation to be performed (add/get/set/delete) as an attribute value for each configurable item.

When exporting the configuration, the xml configuration parameter specifies what portion of the configuration should be exported by including the desired subset of the configuration hierarchy from the root down to the element in the schema to be exported. That element would be marked with a "get" attribute to indicate the export operation. That element may be the root, which would imply export of the complete configuration, or multiple elements below the root may be marked with the "get" attribute to export multiple portions at the same time.

When importing the configuration, the xml configuration parameter would include the portion of the configuration to be changed, with an attribute of "add", "set" or "delete" on the affected elements, and with the element containing the information to be added or set as appropriate.

A single API call is allowed to import some portions of the configuration in the xml configuration input parameter while that parameter also specifies exporting of some other portions so long as those portions are on separate branches of the schema hierarchy.

Obtaining the Schema

For details on the XML format and what operations can be specified with each configurable element, refer to the configuration schema, which can be downloaded by following these steps:

1 Open a web browser, enter the appliance IP address, and log in as prompted to open the web interface.

2 Choose **Support** to open the Support page.

3 In the XML Schema for Configuration Export/Import API section, click **Download XML Schema file**.

4 Save the schema.

api_manage_config

API Call: api_manage_config {password} {user} {xml configuration}
This query supports configuration of elements in the configuration schema.

Input Parameters

- **password**—The password configured for queries.
- **user**—The user who owns the groups to be manipulated.
- **xml configuration—** An XML RPC string that contains all or a portion of the system configuration in XML format encoded with "XML Safe Encoding" so that the XML tags in the configuration are not interpreted as XML RPC types. For example, the XML Safe Encoding converts "<" to "<". The configuration string contains the subset of the configuration schema to be exported or imported. Refer to the subsequent sections in this chapter for examples of specific configuration tasks.

Structure of Output

- vinfo: version struct
- result: string. For an add, delete, or set operation, the string will be empty. If one or more get operations were included in the xml configuration parameter, then the output will be the XML Safe Encoding of the requested configuration in XML format according to the configuration schema. See the sample output for reference.

Example

The following example program is generic for all uses of the API. For a specific task, provide an XML file containing the applicable elements from the schema as a parameter to the program.

```
#!/usr/bin/perl

use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use MIME::Base64;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::Indent = 1;
```

```
if(!defined($ARGV[0]) || !defined($ARGV[1]) || !defined($ARGV[2]) ||
!defined($ARGV[3])) {
    printf "usage: ./api_manage_config.pl ip port password xmlFile\n";
    exit(0);
}

my $rexip = $ARGV[0];
my $port = $ARGV[1];
my $password = $ARGV[2];
my $user = 'admin';
my $xml;
if (defined($ARGV[3])) {
local $/=undef;
open XMLFILE, $ARGV[3] or die "Couldn't open file: $!";
$xml = <XMLFILE>;
close XMLFILE;
$xml =~ s/\s*<\s*/</g;
$xml =~ s/\s*>\s*/>/g;
}

my $client;
my $req;
my @reqs;
my $t1 = `date +%s`;

$client = new RPC::XML::Client "http://$rexip:$port/RPC2";

push (@reqs, RPC::XML::request->new('api_manage_config',
           RPC::XML::RPC_STRING($password),
           RPC::XML::RPC_STRING($user),
           RPC::XML::RPC_STRING($xml))
           );
foreach  (@reqs) {
     my $res = $client->send_request($_);
     if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
     my $value1 = $res->value;
     print Dumper($value1);
}
```

Sample Output

The following sample output is for an XML file containing add, set, or delete operations but no get operations. For sample outputs of get operations, see the subsequent sections on specific configuration tasks. This sample output has been reformatted by the Dumper function.

```
{
  'vinfo' => {
    'software_version' => '9.4.27-E Route Explorer',
    'appliance_version' => '9.4.27'
  },
  'result' => '
'
}
```

Configuring Alerts

For alert configuration, the applicable subset of the following elements should be included in XML format conforming to the schema:

- **PeeringStateAlerts**: array of peering state alerts in XML format conforming to the schema
- **PrefixStateAlerts**: array of prefix state alerts in XML format conforming to the schema
- **AdjStateAlerts**: array of adjacency state alerts in XML format conforming to the schema
- **RouterStateAlerts**: array of router state alerts in XML format conforming to the schema
- **PrefixFloodDroughtAlerts**: array of prefix flood/drought state alerts in XML format conforming to the schema
- **RedundancyAlerts**: array of redundancy state alerts in XML format conforming to the schema
- **ASPathAlerts**: array of AS path alerts in XML format conforming to the schema
- **RouterGroups**: array of router groups in XML format conforming to the schema
- **PathGroups**: array of path groups in XML format conforming to the schema
- **LinkGroups**: array of link groups in XML format conforming to the schema

- **IPv4PrefixGroups**: array of ipv4 prefix groups in XML format conforming to the schema
- **IPv6PrefixGroups**: array of ipv6 prefix groups in XML format conforming to the schema
- **DispatchSpecs**: array of dispatch specifications in XML format conforming to the schema
- **SuppressionSpecs**: array of suppression specifications in XML format conforming to the schema

Example (Path Alert)

This section provides example XML input files for path alerts. A path alert will reference dispatch and suppression specifications plus a path group to list the paths to be monitored.

The following XML input file sets the dispatch and suppression specifications:

```
<Configuration>
<schemaVersion>2.0</schemaVersion>
<DispatchSpecs>
<dispatchSpec operation="add">
<name>apiDispatch</name>
<logToDB>1</logToDB>
<snmp>
<address>
<ipAddress>10.64.15.58</ipAddress>
<port>162</port>
</address>
<community>public</community>
</snmp>
<syslog>
<address>
<ipAddress>10.64.15.58</ipAddress>
<port>514</port>
</address>
<syslogFacility>local6</syslogFacility>
</syslog>
<email>
<from>RouteExplorer@packetdesign.com</from>
<to>qa@qa.packetdesign.com</to>
<mailServer>qa.packetdesign.com</mailServer>
</email>
</dispatchSpec>
</DispatchSpecs>
<SuppressionSpecs>
<suppressionSpec operation="add">
```

```
<name>5PerHour</name>
<rateLimit>
<count>5</count>
<duration>1</duration>
<durationType>hours</durationType>
</rateLimit>
<schedule>
<one_time>
<fromDate>2011-01-01 01:00:00</fromDate>
<toDate>2011-01-01 01:00:00</toDate>
</one_time>
</schedule>
</suppressionSpec>
</SuppressionSpecs>
</Configuration>
```

The following XML input file sets a path group and a path alert that references the dispatch and suppression specifications and path group:

```
<Configuration>
<schemaVersion>2.0</schemaVersion>
<networkWideConfiguration>
<networkName>PDI</networkName>
<PathGroups>
<pathGroup operation="add">
<name> PathGrp1</name>
<user> admin</user>
<path>
<source>
<router>
<ipAddress>3.3.3.3</ipAddress>
</router>
</source>
<destination>
<ipv4addr>
<prefix>6.6.6.6</prefix>
<mask>32</mask>
</ipv4addr>
</destination>
</path>
</pathGroup>
</PathGroups>
```

```
<PathAlerts>
<pathAlert operation="add">
<watchlist>
<name>PathGrp1</name>
<type>Paths Group</type>
</watchlist>
<alertParameters>
<suppressionSpec>5PerHour</suppressionSpec>
<severity>Notice</severity>
<dispatchSpec>apiDispatch</dispatchSpec>
</alertParameters>
<alertOnMetricChange>1</alertOnMetricChange>
<alertOnHopOrECMPDegChange>1</alertOnHopOrECMPDegChange>
<alertOnDelayChange>1</alertOnDelayChange>
<thresholdType>Absolute</thresholdType>
<delayThreshold>1000</delayThreshold>
</pathAlert>
</PathAlerts>
</networkWideConfiguration>
</Configuration>
```

The following XML input file gets back the results of the alert configuration operations:

```
<Configuration>
<schemaVersion>2.0</schemaVersion>
<networkWideConfiguration>
<networkName>PDI</networkName>
<PathGroups operation="get">
</PathGroups>
<PathAlerts operation="get">
</PathAlerts>
</networkWideConfiguration>
<DispatchSpecs operation="get">
</DispatchSpecs>
<SuppressionSpecs operation="get">
</SuppressionSpecs>
</Configuration>
```

Sample Output (Path Alerts)

The following sample output shows the result of the get operation on the configured path alert. The output has been reformatted by the Dumper function, so the XML Safe Encoding of the configuration string has been decoded.

```
{
  'vinfo' => {
    'software_version' => '9.4.27-E Route Explorer',
    'appliance_version' => '9.4.27'
  },
  'result' => '
<Configuration>
 <schemaVersion>
  1.0
 </schemaVersion>
 <networkWideConfiguration>
  <networkName>
   PDI
  </networkName>
  <PathGroups>
   <pathGroup>
    <name>
     PathGrp1
    </name>
    <user>
     admin
    </user>
    <path>
     <source>
      <router>
       <routerName>
        P3
       </routerName>
       <ipAddress>
        3.3.3.3
       </ipAddress>
      </router>
     </source>
```

```
    <destination>
     <ipv4addr>
      <prefix>
       6.6.6.6
      </prefix>
      <mask>
       32
      </mask>
     </ipv4addr>
    </destination>
   </path>
  </pathGroup>
</PathGroups>
<PathAlerts>
 <pathAlert>
  <watchlist>
   <name>
    PathGrp1
   </name>
   <type>
    Paths Group
   </type>
  </watchlist>
  <alertParameters>
   <dispatchSpec>
    apiDispatch
   </dispatchSpec>
   <suppressionSpec>
    5PerHour
   </suppressionSpec>
   <severity>
    Notice
   </severity>
  </alertParameters>
  <alertOnMetricChange>
   true
  </alertOnMetricChange>
  <alertOnHopOrECMPDegChange>
   true
  </alertOnHopOrECMPDegChange>
  <alertOnDelayChange>
   true
```

```
    </alertOnDelayChange>
    <thresholdType>
     Absolute
    </thresholdType>
    <delayThreshold>
     1000
    </delayThreshold>
   </pathAlert>
  </PathAlerts>
</networkWideConfiguration>
<DispatchSpecs>
 <dispatchSpec>
  <name>
   apiDispatch
  </name>
  <logToDB>
   0
  </logToDB>
  <snmp>
   <address>
    <ipAddress>
     10.64.15.58
    </ipAddress>
    <port>
     162
    </port>
   </address>
   <community>
    public
   </community>
  </snmp>
  <syslog>
   <address>
    <ipAddress>
     10.64.15.58
    </ipAddress>
    <port>
     514
    </port>
   </address>
   <syslogFacility>
    local6
```

```
    </syslogFacility>
   </syslog>
   <email>
    <from>
     RouteExplorer@packetdesign.com
    </from>
    <to>
     qa@qa.packetdesign.com
    </to>
    <mailServer>
     qa.packetdesign.com
    </mailServer>
   </email>
  </dispatchSpec>
 </DispatchSpecs>
 <SuppressionSpecs>
  <suppressionSpec>
   <name>
    5PerHour
   </name>
   <rateLimit>
    <count>
     5
    </count>
    <duration>
     1
    </duration>
    <durationType>
     hours
    </durationType>
   </rateLimit>
   <schedule>
    <one_time>
     <fromDate>
      2011-01-01 01:00:00
     </fromDate>
     <toDate>
      2011-01-01 01:00:00
     </toDate>
    </one_time>
   </schedule>
  </suppressionSpec>
```

```
 </SuppressionSpecs>
</Configuration>
'
}
```

A Deprecated IP Alert Queries

This appendix describes the deprecated queries for IP alert configuration. These queries are still functional in this release, but will be removed in a future release. These queries have been superseded by the api_manage_config query that will eventually allow configuration of all aspects of the appliance operation, not just alerts. See Chapter 7, "Configuration Queries"

The parameters of a IP alert include a group, to specify a list of objects to be watched; a dispatch specification, to tell how the alert should be delivered; and an optional suppression specification, to limit the alert rate or the time periods in which alerts can be sent. Two queries are provided to add or delete any combination of these parameters along with the alerts that reference them. A third query reads back all of the ip alerts and their parameters. The client application can display the ip alerts and parameters configured through the XML RPC API. Conversely, the queries can reference or add to parameter specifications created using the client application and will read back the ip alerts, groups, dispatch specifications, and suppressions specifications created by the client application.

For additional information on IP alerts and the information carried in the dispatch and suppression specifications, see the "Alerts" chapter in the *HP Route Analytics Management System User's Guide*.

Query Data Structures

Several data structures are used for the configuration input parameters and output results in the different IP alert queries. At the top level is an XML RPC struct that can contain all or any portion of the alert configuration. The members of the struct vary according to which alerts are configured. All of the alerts share dispatch and suppression specifications.

Dispatch Specifications

The dispatch array in the configuration struct input parameter for api_add_config and api_delete_config contains one or more dispatch specifications. Each dispatch specification can specify any combination of the delivery protocols snmp, syslog and/or email, plus an option to log to an internal alert database. Each dispatch specification has a name to allow alert configurations to refer to it. Use the following struct members to define a dispatch specification:

- name: string (used to identify the dispatch specification)
- log_to_db: boolean (“true” or “false”)
- snmp: array of SNMP dispatch specification structs (optional)
  - address: string (SNMP server IP address or hostname)
  - port: int (SNMP server listening port)
  - community: string (optional SNMP community identifier)
- syslog: array of syslog dispatch specification structs (optional)
  - address: string (syslog server IP address or hostname)
  - port: int (syslog server listening port)
  - syslog_facility: string “local0” - “local7” (optional, sets syslog facility identifier for all alerts)
- email: array of email dispatch specification structs (optional)
  - from: string (email address for “From:” field)
  - to: array of strings (recipient email addresses)
  - mail_server: string (optional, sets email server name or address to use for all alerts)

Suppression Specifications

The suppression array in the configuration struct input parameter for api_add_config and api_delete_config contains one or more suppression specifications. Each suppression specification can specify one or both methods of alert suppression.

The first method is to define a time period during which alerts are completely suppressed. That period is defined by a start and end time and may occur just once or be repeated on a daily, weekly, monthly or yearly basis, or a multiple thereof. The second method is to impose a rate limit on dispatching of alerts that applies at all times. The rate limit is expressed as a maximum count of alerts allowed to be sent within a period of a specified duration. Each suppression specification has a name to allow alert configurations to refer to it. Use the following struct members to define a suppression specification:

- name: string (used to identify the suppression specification)
- event_type: string, one of “one_time”, “daily”, “weekly”, “monthly”, or “yearly”
- start: dateTime.iso8601, a UTC time in ISO 8601 format to start suppressing notifications
- end: dateTime.iso8601, a UTC time in ISO 8601 format to stop suppressing notifications
- interval: int, the multiple of the event_type interval at which the suppression period is repeated
- count: int, the maximum count of alerts allowed in the rate limit period
- duration: int, the number of seconds in the rate limit period

Path Alert

For a path alert, the top level stuct includes any combination of the following components:

- path_groups: array of path group structs
- dispatch: array of dispatch specification structs
- suppression: array of suppression specification structs
- path_alerts: array of path alert structs

The next sections describe each of these structs.

Path Groups

The path_groups array in the configuration struct input parameter for api_add_config and api_delete_config contains one or more path group specifications. Each path group contains a list of paths identified by a router address as the source and a prefix as the destination. The source address can be a router ID or an interface address. The destination prefix can be known in the network topology, or not. In the latter case, the path extends to an exit router. The path is routed using IPv4 or IPv6 according to whether the source and destination addresses are contained in an ip4_addr or ip6_addr string.

Each path group has a name to allow alert configurations to refer to it. Path groups can also be constructed hierarchically such that one path group refers to other path groups as children. Use the following struct members to define a path group:

- name: string (used to identify the path group)
- user: string (the account that will own the group)
- paths: array of path structs (optional)
  - source: IP struct containing one of the following:
    - ip4_addr: string (address of the source router)
    - ip6_addr: string (address of the source router)
  - destination: prefix struct
    - masklen: int
    - ip_addr: IP struct containing one of the following:
      - ip4_addr: string
      - ip6_addr: string
- children: array of strings (optional names of child path groups)

Path Alert Configurations

The path_alerts array in the configuration struct input parameter for api_add_config and api_delete_config contains specifications for one or more path alerts. A prerequisite to configuring a path alert is that a path group and dispatch specification must exist, and optionally a suppression specification may be referenced. The path alert configuration takes one additional parameter that specifies the severity of the alert.

Alerts are not named, so to delete a path alert requires specifying the same parameters (by name) as when the alert was created in order to identify the alert. Use the following struct members to define a path alert:

- severity: string, one of Info, Notice, Warning, Error, or Critical
- watchlist: string referencing the path group name
- dispatch: string referencing a dispatch specification
- suppression: string referencing a suppression specification

Prefix State Alert

For a prefix state alert, the top level stuct includes any combination of the following components:

- ipv4/ipv6: array of group structs
- dispatch: array of dispatch specification structs
- suppression: array of suppression specification structs
- prefix_alerts: array of prefix state alert structs

Prefix Groups

The ipv4_prefix_groups and ipv6_prefix_groups arrays in the configuration struct input parameter for api_add_config and api_delete_config contains one or more ipv4/ipv6 specifications. Each prefix group contains a list of prefixes and also a name to allow alert configurations to refer to it. Prefix groups can also be constructed hierarchically such that one prefix group refers to other prefix groups as children. Use the following struct members to define a prefix group:

- name: string (used to identify the prefix group)
- user: string (the account that will own the group)
- prefixes: array of prefix structs
  - masklen: int
  - ip_addr: IP struct containing one of the following:
    - ip4_addr: string
    - ip6_addr: string

- children: array of strings (optional names of child prefix groups)

Prefix State Alert Configurations

The prefix_alerts array in the configuration struct input parameter for api_add_config and api_delete_config contains specifications for one or more prefix state alerts. A prerequisite to configuring a prefix state alert is that a prefix group and dispatch specification must exist, and optionally a suppression specification may be referenced.

The prefix state alert configuration takes two additional parameters, one that specifies the severity of the alert and other the alert condition.

Alerts are not named, so to delete a prefix state alert requires specifying the same parameters (by name) as when the alert was created in order to identify the alert.

Use the following struct members to define a prefix alert:

- severity: string, one of Info, Notice, Warning, Error, or Critical
- watchlist: string referencing the prefix group name
- dispatch: string referencing a dispatch specification
- suppression: string referencing a suppression specification
- alert_condition: string, one of Up, Down, Flap.
- flap_count: int, number of flaps within the specified duration.
- duration: int (In seconds)

Adjacency State Alert

For an adjacency state alert, the top level stuct includes any combination of the following components:

- link_groups: array of link group structs
- dispatch: array of dispatch specification structs
- suppression: array of suppression specification structs
- adjacency_alerts: array of adjacency state alert structs

Link Groups

The link_groups array in the configuration struct input parameter for api_add_config and api_delete_config contains one or more link_groups specifications. Each link group contains a list of links and also a name to allow alert configurations to refer to it. Link groups can also be constructed hierarchically such that one link group refers to other link groups as children. Use the following struct members to define a link group:

- name: string (used to identify the link group)
- user: string (the account that will own the group)
- links: array of link structs
  - srcNode: MP router struct containing one of the following:
    - name: string (name of the source router)
    - ip4_addr: string (address of the source router)
  - dstNode: MP router struct containing one of the following:
    - name: string (name of the source router)
    - ip4_addr: string (address of the source router)
- children: array of strings (optional names of child link groups)

Adjacency Alert Configurations

The adjacency_alerts array in the configuration struct input parameter for api_add_config and api_delete_config contains specifications for one or more adjacency state alerts. A prerequisite to configuring a adjacency alert is that a link group and dispatch specification must exist, and optionally a suppression specification may be referenced. The adjacency state alert configuration takes two additional parameter one that specifies the severity of the alert and other is alert condition specification.

Alerts are not named, so to delete a adjacency state alert requires specifying the same parameters (by name) as when the alert was created in order to identify the alert.

Use the following struct members to define a adjacency alert:

- severity: string, one of Info, Notice, Warning, Error, or Critical
- watchlist: string referencing the link group name

- dispatch: string referencing a dispatch specification
- suppression: string referencing a suppression specification
- alert_condition: string, one of Up, Down, Flap.
- flap_count: int, number of flaps within the specified duration.
- duration: int (In seconds)

Router State Alerts

For a router state alert, the top level stuct includes any combination of the following components:

- router_groups: array of router group structs
- dispatch: array of dispatch specification structs
- suppression: array of suppression specification structs
- router_alerts: array of router state alert structs

Router Groups

The router_groups array in the configuration struct input parameter for api_add_config and api_delete_config contains one or more router groups specifications. Each router group contains a list of routers and also a name to allow alert configurations to refer to it. Router groups can also be constructed hierarchically such that one router group refers to other router groups as children. Use the following struct members to define a router group:

- name: string (used to identify the router group)
- user: string (the account that will own the group)
- routers: array of router structs (optional)
  - name: string (name of the source router)
  - ip4_addr: string (address of the source router)
- children: array of strings (optional names of child router groups)

Router State Alert Configurations

The router_alerts array in the configuration struct input parameter for api_add_config and api_delete_config contains specifications for one or more router state alerts. A prerequisite to configuring a router state alert is that a router group and dispatch specification must exist, and optionally a suppression specification may be referenced. The router state alert configuration takes two additional parameter one that specifies the severity of the alert and other is alert condition specification.

Alerts are not named, so to delete a adjacency alert requires specifying the same parameters (by name) as when the alert was created in order to identify the alert. Use the following struct members to define a router state alert:

- severity: string, one of Info, Notice, Warning, Error, or Critical
- watchlist: string referencing the router group name
- dispatch: string referencing a dispatch specification
- suppression: string referencing a suppression specification
- alert_condition: string, one of Isolation, Connection, Flap.
- flap_count: int, number of flaps within the specified duration.
- duration: int (In seconds)

Queries

The remainder of this appendix describes the api_add_config, api_delete_config and api_get_config queries. Each requires a database name and a time specification as input parameters.

The database name should be the top-level administrative domain name in the hierarchy that describes the network topology in the system. The alert configuration is associated with the entire topology, not just a portion of it, although the trigger for any particular alert may be specific to just one element in the network.

The exact value chosen for the time parameter is not critical. It is typically near the current time of day (in the UTC time zone). The only effect of the time parameter is to load the state of the network for the purpose of finding routers when referenced by address. If the set of routers in the network is stable, then

all times within the period of stability are equivalent. On the other hand, if you need to configure a path alert for a router that does not exist at the current time, but did exist previously and will exist again in the future, then select a time when that router did exist.

To issue several path alert queries in conjunction, supply the same time parameter value on all of those queries. That will allow the Query Server to load the state of the network just once and then reference that same loaded network topology to respond to each query.

It may be convenient to automatically fetch the current time and supply it as the time parameter when each query is issued, but in that case the server must reload the network topology for each query. An alternative solution that retains this convenience is to automatically fetch the current time but then truncate the time back to the last five minute boundary (:00, :05, :10, etc.). That way the Query Server needs to reload the network topology at most once every five minutes.

Alerts.api_add_config

API Call: Alerts.api_add_config {password} {database name} {time} {configuration struct}

This query creates or adds parameters into any combination of groups, dispatch specifications, suppression specifications, and alert configurations. These elements are all combined into one XML RPC struct that is input as the fourth parameter in the query. When creating an alert, the group, dispatch specification and suppression specification (if used) that are referenced by the alert must either be created in the same query or in an earlier query or through the client application.

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which should be the top-level administrative domain for the network, such as CorpNet.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query finds the source router for a path based on the network state at the specified time.

- **configuration struct —** An XML RPC struct containing any combination of the following members:
  - **path_alerts**: array of path alert structs
  - **path_groups**: array of path group structs
  - **prefix_alerts**: array of prefix state alert structs
  - **ipv4_prefix_groups**: array of ipv4 prefix group structs
  - **ipv6_prefix_groups**: array of ipv6 prefix group structs
  - **adjacency_alerts**: array of adjacency state alert structs
  - **link_groups**: array of link group structs
  - **router_alerts**: array of router state alert structs
  - **router_groups**: array of router group structs
  - **dispatch**: array of dispatch specification structs
  - **suppression**: array of suppression specification structs

Structure of Output

- vinfo: version struct

Example (Path Alert)

```
#!/usr/bin/perl
my $ip = "10.64.15.219";
my $database = "dedupDB";
my $port = 2000;

use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$ip:$port";

my $t1 = str2time("20090310T15:30:00");
```

```
my $t2 = str2time("20090312T10:00:00");

my $paths = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'source' => RPC::XML::struct->new(
                    'ip4_addr' =>
RPC::XML::string->new('10.130.1.28')
                  ),
                  'destination' => RPC::XML::struct->new(
                    'ip_addr' => RPC::XML::struct->new(
                      'ip4_addr' =>
RPC::XML::string->new('10.71.2.21')
                    ),
                    'masklen' => RPC::XML::int->new(32)
                  )
                ),
                RPC::XML::struct->new(
                  'source' => RPC::XML::struct->new(
                    'ip4_addr' =>
RPC::XML::string->new('10.130.1.25')
                  ),
                  'destination' => RPC::XML::struct->new(
                    'ip_addr' => RPC::XML::struct->new(
                      'ip4_addr' =>
RPC::XML::string->new('10.130.1.28')
                    ),
                    'masklen' => RPC::XML::int->new(32)
                  )
                )
);

my $child = RPC::XML::array->new(
                RPC::XML::string->new("az"),
                RPC::XML::string->new("bb")
);

my $path_group = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'user' => RPC::XML::string->new('admin'),
                 'name' => RPC::XML::string->new('PathGrp1'),
                  'paths' => $paths,
                  'children' => $child
```

```
                )
);


my $snmp = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'address'
=> RPC::XML::string->new('10.64.15.99'),
                  'port' => RPC::XML::int->new(162),
                  'community' =>
RPC::XML::string->new('public')
                ),
);

my $syslog = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'address' =>
RPC::XML::string->new('10.64.15.99'),
                  'port' => RPC::XML::int->new(514),
                  'syslog_facility'=> RPC::XML::string->new('
local0')
                ),
);

my $email = RPC::XML::struct->new(
                'from' =>
RPC::XML::string->new('route_recorder@pd.com'),
                'to' => RPC::XML::array->new(
                  RPC::XML::string->new('support_level_one@pd
.com'),
                  RPC::XML::string->new('dispatch_level_two@p
d.com')
                ),
                'mail_server' =>
RPC::XML::string->new('california.pd.com')
);

my $array_dispatch = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'name' =>
RPC::XML::string->new('dispatch_set_one'),
```

```
                  'log_to_db' =>
RPC::XML::boolean->new('false'),
                  'snmp' => $snmp,
                  'syslog' => $syslog,
                  'email' => $email
                )
);

my $supp_specs = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'name' =>
RPC::XML::string->new('suppress_one'),
                  'event_type' =>
RPC::XML::string->new('one_time'),
                  'start' =>
RPC::XML::datetime_iso8601->new($t1),
                  'end' =>
RPC::XML::datetime_iso8601->new($t2),
                ),
                RPC::XML::struct->new(
                  'name' =>
RPC::XML::string->new('suppress_two'),
                  'event_type' =>
RPC::XML::string->new('monthly'),
                  'start' =>
RPC::XML::datetime_iso8601->new($t1),
                  'end' =>
RPC::XML::datetime_iso8601->new($t2),
                  'interval' => RPC::XML::int->new(2),
                  'count' => RPC::XML::int->new(5),
                  'duration' => RPC::XML::int->new(10)
                )
);

my $alerts = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'severity' =>
RPC::XML::string->new('Notice'),
                  'watchlist' =>
RPC::XML::string->new('PathGrp1'),
                  'dispatch' =>
RPC::XML::string->new('dispatch_set_one'),
```

```
                    'suppression' =>
RPC::XML::string->new('suppress_one')
                  )
);

my $config = RPC::XML::struct->new(
                  'path_groups' => $path_group,
                  'dispatch' => $array_dispatch,
                  'suppression' => $supp_specs,
                  'path_alerts' => $alerts
);

push (@reqs, RPC::XML::request->new('Alerts.api_add_config',
             RPC::XML::RPC_STRING($password),
             RPC::XML::RPC_STRING($database),

          RPC::XML::datetime_iso8601->new(time2iso8601($t1)))
             $config)
);

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;
        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' =>  '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  }
}
```

Alerts.api_delete_config

API Call: Alerts.api_delete_config {password} {database name} {configuration structure} This query deletes any combination of groups, dispatch specifications, suppression specifications, and alert configurations. Also, this query will allow the user to delete specific group members passed in configuration struct.

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which should be the top-level administrative domain for the network, such as CorpNet.
- **time** —A time specified in ISO 8601 format in the UTC time zone, such as 20050725T21:47:35. The query finds the source router for a path based on the network state at the specified time.
- **configuration struct**—Similar to api_add_config, except that this query deletes only the leaves of a specified struct. To delete a full path group, for example, specify only the path group name, not the path or children.

Structure of Output

- vinfo: version struct

Example (Path Alert)

```
#!/usr/bin/perl
my $ip = "10.64.15.219";
my $database = "dedupDB";
my $port = 2000;

use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
```

```
$client = new RPC::XML::Client "http://$ip:$port";

my $t1 = str2time("20090310T15:30:00");
my $t2 = str2time("20090312T10:00:00");

my $array_dispatch = RPC::XML::array->new(
                RPC::XML::struct->new(
                  'name' =>
RPC::XML::string->new('dispatch_set_one')
                )
);

my $supp_specs = RPC::XML::array->new(
                RPC::XML::struct->new(
               'name' => RPC::XML::string->new('suppress_one')
                ),
                RPC::XML::struct->new(
               'name' => RPC::XML::string->new('suppress_two')
                )
);

my $alerts = RPC::XML::array->new(
                RPC::XML::struct->new(
               'severity' => RPC::XML::string->new('Notice'),
                  'watchlist' =>
RPC::XML::string->new('PathGrp1'),
                  'dispatch' =>
RPC::XML::string->new('dispatch_set_one'),
                  'suppression'
=> RPC::XML::string->new('suppress_one')
                )
);

my $config = RPC::XML::struct->new(
                'dispatch' => $array_dispatch,
                'suppression' => $supp_specs,
                'path_alerts' => $alerts
);

push (@reqs,
RPC::XML::request->new('Alerts.api_delete_config',
           RPC::XML::RPC_STRING($password),
```

```
            RPC::XML::RPC_STRING($database),

          RPC::XML::datetime_iso8601->new(time2iso8601($t1)))
            $config)
);

foreach (@reqs) {
        my $res = $client->send_request($_);
        if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
        my $value1 = $res->value;
        print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  }
}
```

•

Alerts.api_get_config

API Call: Alerts.api_get_config {password} {database name} {time}

This query returns all of the currently configured alerts, groups, dispatch specifications and suppression specifications.

Input Parameters

- **password**—The password configured for queries.
- **database name**—A name from the database hierarchy, which should be the top-level administrative domain for the network, such as CorpNet.
- **time —**A time specified in ISO 8601 format in the UTC time zone, such as `20050725T21:47:35`. The query results are assembled based on the network state at the specified time

Structure of Output

- vinfo: version struct
- path_groups: array of path group structs, if any, as follows:
  - name: string
  - user: string
  - paths: array of path structs, if any
    - source: IP struct containing one of the following:
      - ip4_addr: string
      - ip6_addr: string
    - destination: prefix struct
      - masklen: int
      - ip_addr: IP struct containing one of the following:
      - ip4_addr: string
      - ip6_addr: string
  - children: array of strings if any child path groups are configured
- prefix_groups: array of prefix groups, if any, as follows:
  - name: string (used to identify the prefix group)

- user: string (the account that will own the group)
- prefixes: array of prefix structs (optional)
  - source: IP struct containing one of the following:
    - ip4_addr: string (address of the source router)
    - ip6_addr: string (address of the source router)
  - destination: prefix struct
    - masklen: int
    - ip_addr: IP struct containing one of the following:
    - ip4_addr: string
    - ip6_addr: string
- children: array of strings (optional names of child prefix groups)

- link_groups: array of link group structs, if any, as follows:
  - name: string (used to identify the link group)
  - user: string (the account that will own the group)
  - links: array of link structs
    - srcNode: MP router struct containing one of the following:
      - name: string (name of the source router)
      - ip4_addr: string (address of the source router)
    - dstNode: MP router struct containing one of the following:
      - name: string (name of the source router)
      - ip4_addr: string (address of the source router)
  - children: array of strings (optional names of child link groups)
- router_groups: Array of router group structs, if, any, as follows:
  - name: string (used to identify the router group)
  - user: string (the account that will own the group)
  - routers: array of router structs (optional)
    - name: string (name of the source router)
    - ip4_addr: string (address of the source router)
  - children: array of strings (optional names of child router groups)

- dispatch: array of dispatch specification structs, as follows:
  - name: string
  - log_to_db: boolean
  - snmp: array of SNMP dispatch specification structs, if any
    - address: string
    - port: int
    - community: string
  - syslog: array of syslog dispatch specification structs, if any
    - address: string
    - port: int
    - syslog_facility: string “local0” - “local7”
  - email: array of email dispatch specification structs, if any
    - from: string
    - to: array of strings
    - mail_server: string
- suppression: array of suppression specification structs, if any, as follows:
  - name: string
  - event_type: string, one of “one_time”, “daily”, “weekly”, “monthly”, or “yearly”
  - start: dateTime.iso8601
  - end: dateTime.iso8601
  - interval: int
  - count: int
  - duration: int
- path_alerts: array of path alert structs, as follows:
  - severity: string, one of “Info”, “Notice”, “Warning”, “Error” or “Critical”
  - watchlist: string
  - dispatch: string

  - suppression: string
- prefix_alerts: array of prefix alert structs, as follows:
  - severity: string, one of Info, Notice, Warning, Error, or Critical
  - watchlist: string referencing the prefix group name
  - dispatch: string referencing a dispatch specification
  - suppression: string referencing a suppression specification
  - alert_condition: string, one of Up, Down, Flap.
  - flap_count: int, number of flaps within the specified duration.
  - duration: int (In seconds)
- adjacency_alerts: array of adjacency alert structs, as follows:
  - severity: string, one of Info, Notice, Warning, Error, or Critical
  - watchlist: string referencing the link group name
  - dispatch: string referencing a dispatch specification
  - suppression: string referencing a suppression specification
  - alert_condition: string, one of Up, Down, Flap.
  - flap_count: int, number of flaps within the specified duration.
  - duration: int (In seconds)
- router_alerts: array of router alert structs, as follows:
  - severity: string, one of Info, Notice, Warning, Error, or Critical
  - watchlist: string referencing the router group name
  - dispatch: string referencing a dispatch specification
  - suppression: string referencing a suppression specification
  - alert_condition: string, one of Isolation, Connection, Flap.
  - flap_count: int, number of flaps within the specified duration.
  - duration: int (In seconds)

Example

```
#!/usr/bin/perl
my $qsip = "10.64.15.219";
```

```
my $database = "dedupDB";
my $filter = "any";
use strict;
use RPC::XML::Client;
use RPC::XML 'time2iso8601';
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1;
$Data::Dumper::Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'packet';
$client = new RPC::XML::Client "http://$qsip:2000/RPC2";

my $t1 = str2time("20090220T15:30:00");

push (@reqs, RPC::XML::request->new('Alerts.api_get_config',
          RPC::XML::RPC_STRING($password),
          RPC::XML::RPC_STRING($database),
          RPC::XML::datetime_iso8601->new(time2iso8601($t1)))
          );

foreach (@reqs) {
     my $res = $client->send_request($_);
     if ($res->is_fault) {print("---XMLRPC FAULT ---"); }
     my $value1 = $res->value;

     print Dumper($value1);
}
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '8.0.30-R Traffic Explorer',
    'appliance_version' => '8.0.30-R'
  },
  'path_alerts' => [
    {
      'dispatch' => 'dispatch_set_one',
      'suppression' => 'suppress_one',
      'severity' => 'Notice',
```

```
    'watchlist' => 'PathGrp1'
  }
],
'dispatch' => [
  {
    'snmp' => [
      {
        'address' => '10.64.15.99',
        'port' => '162',
        'community' => 'public'
      }
    ],
    'email' => {
      'to' => [
        'support_level_one@pd.com',
        'dispatch_level_two@pd.com'
      ],
      'mail_server' => 'california.pd.com',
      'from' => 'route_recorder@pd.com'
    },
    'name' => 'dispatch_set_one',
    'syslog' => [
      {
        'syslog_facilty' => 'local0',
        'address' => '10.64.15.99',
        'port' => '514'
      }
    ],
    'log_to_db' => 0
  }
'path_groups' => [
  {
    'name' => 'PathGrp1',
    'children' => [
      'bb',
      'az'
    ],
    'paths' => [
      {
        'source' => {
          'ip4_addr' => '10.130.1.25'
        },
```

```
      'destination' => {
        'masklen' => '32',
        'ip_addr' => {
          'ip4_addr' => '10.130.1.28'
        }
      }
    },
    {
      'source' => {
        'ip4_addr' => '10.130.1.28'
      },
      'destination' => {
        'masklen' => '32',
        'ip_addr' => {
          'ip4_addr' => '10.71.2.21'
        }
      }
    }
  ]
},
{
  'name' => 'az',
  'children' => [],
  'paths' => []
},
{
  'name' => 'bb',
  'children' => [],
  'paths' => [
    {
      'source' => {
        'ip4_addr' => '10.40.0.1'
      },
      'destination' => {
        'masklen' => '32',
        'ip_addr' => {
          'ip4_addr' => '10.2.0.1'
        }
      }
    },
    {
      'source' => {
```

```
            'ip4_addr' => '10.40.0.1'
          },
          'destination' => {
            'masklen' => '24',
            'ip_addr' => {
              'ip4_addr' => '10.2.1.0'
            }
          }
        },
        {
          'source' => {
            'ip4_addr' => '10.40.0.1'
          },
          'destination' => {
            'masklen' => '24',
            'ip_addr' => {
              'ip4_addr' => '10.3.0.0'
            }
          }
        },
        {
          'source' => {
            'ip4_addr' => '10.40.0.1'
          },
          'destination' => {
            'masklen' => '24',
            'ip_addr' => {
              'ip4_addr' => '10.10.3.0'
            }
          }
        }
      ]
    }
  ],
  'suppression' => [
    {
      'count' => '5',
      'event_type' => 'weekly',
      'name' => 'suppress_one',
      'duration' => '300',
      'interval' => '1',
      'start' => '20090308T00:02:00',
```

```
      'end' => '20090308T00:03:00'
    }
  ]
}
```

B Deprecated XML RPC Queries

This appendix lists the queries that have been deprecated. These queries are no longer in use, and replacement API's (including their re-entrant version, if available) are listed below.

For information about re-entrant queries, see Chapter 3, "Using Re-Entrant Queries"

Deprecated Queries

Table A-1 lists the deprecated queries for this release.

Table 4 Deprecated XML RPC Queries

Deprecated Query	Replacement Query	Re-Entrant Replacement
api_link_list	api_mp_links	n/a
api_list_a_route	api_mp_list_paths	n/a
api_list_a_route_ECMP	api_mp_list_paths	n/a
api_prefix_events	api_mp_events	api_mp_events_handle
api_prefix_list	api_mp_routes	n/a
api_prefix_list_filtered	api_mp_routes	api_mp_routes_handle
api_prefix_multi_orig	api_mp_prefix_multi_orig	n/a
api_router_events	api_mp_events	n/a
api_router_list	api_mp_routers	n/a