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 Notices

Trademark Notices

Linux is a U.S. Registered trademark of Linus Torvalds.

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

Unix® is a registered trademark of The Open Group.

Documentation Updates

This guide's title page 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

You can visit the HP Software Support web site at:

http://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 Support Online 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 HP Software 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 find more information about access levels, go to:

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

To register for an HP Passport ID, go to:

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

Contents

1 Configuring and Using Queries 9
Configuring Route Analytics Management System to Accept Queries 9
Encrypting the API Password 11
Using Queries 12
Understanding Fault Codes 20
Query Data Structures 22
Non-MP/VPN Data Structures 22
MP/VPN Data Structures 24

2 High-Performance Interaction with the Query Server 27
Using a Direct TCP Connection 27
Keeping the Connection Open 29
Cutting Down XML RPC Overhead 31
Concurrency 32
Blocking Queries 34
Handling Topology and Time Changes 34

3 Using Re-Entrant Queries 35

4 XML RPC Queries 39
api_mp_close_handle 39
api_mp_events 41
api_mp_events_handle 47
api_mp_ipv6_routes 49
api_mp_ipv6_routes_handle 53
api_mp_links 56
api_mp_list_handle 60
api_mp_list_paths 62

api_mp_osi_routes . . . 76
api_mp_osi_routes_handle . . . 81
api_mp_prefixes_multi_origin . . . 82
api_mp_prefixes_multi_origin_handle . . . 88
api_mp_routers . . . 90
api_mp_routers_consolidated . . . 94
api_mp_routers_consolidated_handle . . . 100
api_mp_routes . . . 102
api_mp_routes_handle . . . 108
api_prefix_list_multi_orig . . . 110
api_resource_status . . . 114
api_router_summarizable . . . 117
api_system_health . . . 120
api_unit_health . . . 124
api_vpn_cust_rt_list . . . 128
api_vpn_customer_pe_participation . . . 131
api_vpn_customer_pe_list . . . 136
api_vpn_customer_reachability . . . 139
api_vpn_customer_reachability_by_peer . . . 143
api_vpn_route_target_pe_participation . . . 147
api_vpn_route_target_pe_list . . . 153
api_vpn_route_target_reachability . . . 158
api_vpn_route_target_reachability_by_peer . . . 163
api_vpn_routes . . . 168
api_vpn_routes_handle . . . 173

5 VPN Customer Report Queries . . . 175

api_traffic_vpn_customer . . . 175
api_traffic_vpn_customer_cos . . . 180
api_traffic_vpn_customer_cos_history . . . 184
api_traffic_vpn_customer_history . . . 191
api_traffic_vpn_customer_wan_connection . . . 195
api_traffic_vpn_customer_wan_connection_cos . . . 200
api_traffic_vpn_customer_wan_connection_cos_history . . . 205

api_traffic_vpn_customer_wan_connection_history 209
api_traffic_vpn_customer_wan_connection_to_wan_connection 213
api_traffic_vpn_customer_wan_connection_to_wan_connection_history 217
api_traffic_vpn_customer_wan_connection_topn_convers 222
api_traffic_vpn_customer_wan_connection_topn_dsts 226
api_traffic_vpn_customer_wan_connection_topn_port_protocol 230
api_traffic_vpn_customer_wan_connection_topn_srcs 234
api_vpn_customer_default_reporting_wan_connections_get 238
api_vpn_customer_default_reporting_wan_connections_set 241
api_vpn_customer_wan_connection_get_config 244
api_vpn_customer_wan_connection_set_config 251
api_vpn_enabled_customer_list_get_config 254

A Deprecated XML RPC Queries 257
Deprecated Queries 257

Index 259

1 Configuring and Using Queries

This chapter describes how to create Route Analytics Management System queries using an Application Programming Interface (API). Route Analytics Management System Queries are specific queries that 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 from the Route Analytics Management System.
- Integrate Route Analytics Management System 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 Route Analytics Management System 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 RAMS or centralized Modeling Engine 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**.

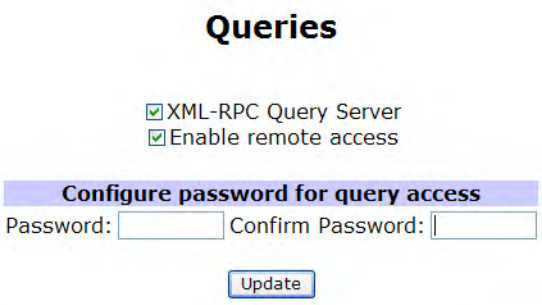

Figure 1 Queries Page

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 remainder 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	4-39
api_mp_events	Lists all multi-protocol network events between two different times.	4-41
api_mp_events_handle	Lists all multi-protocol network events between two different times in chunks of a user-specified size.	4-47
api_mp_links	Lists links meeting filter criteria in a multi-protocol network.	4-56
api_mp_list_handle	Returns a user-specified number of entries starting at a user-specified point in the report	4-60
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.	4-62
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.	4-76

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

Query	Description	Page
api_mp_osi_routes_handle	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.	4-81
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.	4-100
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.	4-81
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).	4-82
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).	4-88

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

Query	Description	Page
api_mp_routes	Lists prefix routes meeting filter criteria in a multi-protocol network.	4-102
api_mp_routes_handle	List prefix routes meeting filter criteria in a multi-protocol network in chunks of a user-specified size.	4-108
api_prefix_list_multi_orig	Lists routers meeting filter criteria in a multi-protocol network.	4-110
api_prefix_list_multi_orig	This query returns a list of prefixes for the specified network that are originated by more than one router.	4-110
api_resource_status	Lists the current status of the memory, disk, and swap space on the appliance.	4-114
api_router_summarizable	Lists routers advertising multiple summarizable prefixes.	4-117
api_system_health	Lists the health of all the RAMS or RAMS Traffic systems in the network, including the recording and writing status of each configured recording process and its databases.	4-120
api_unit_health	Lists the health of the specified RAMS or RAMS Traffic 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.	4-124
api_vpn_cust_rt_list	Lists all customer names in VPN to RT (Route Target) mappings.	4-128
api_vpn_customer_pe_participation	Return statistics of participating PEs for each VPN customer.	4-131

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

Query	Description	Page
api_vpn_customer_pe_list	Return the list of participating PEs for the specified VPN customer.	4-136
api_vpn_customer_reachability	This query returns reachability statistics for each VPN customer.	4-139
api_vpn_route_target_reachability_by_peer	Return reachability statistics at each PE for the specified VPN customer.	4-143
api_vpn_route_target_pe_participation	Return statistics of participating PEs for each route target in the specified network.	4-147
api_vpn_route_target_pe_list	This query returns the list of participating PE routers and their VPN state for the specified route target.	4-153
api_vpn_route_target_reachability	Return reachability statistics for each route target in the specified network.	4-158
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.	4-168
api_vpn_routes_handle	Lists all prefixes advertised in the network.	4-173

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.	5-175
api_traffic_vpn_customer_cos	This query returns breakdown of the aggregate traffic statistics by CoS group.	5-180
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.	5-184
api_traffic_vpn_customer_history	This query returns the history statistics for the VPN customer for the given time period.	5-191
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.	5-195
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.	5-200
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.	5-205
api_traffic_vpn_customer_wan_connection_history	This query retrieves the configuration of the WAN connections for a specific customer.	5-209
api_traffic_vpn_customer_wan_connection_to_wan_connection	This query returns statistics for VPN traffic between the 100 selected WAN connections.	5-213

Table 2 VPN Customer Reports Query Calls

Query	Description	Page
api_traffic_vpn_customer_wan_connection_to_wan_connection_history	This query returns teh history of ingress or egress statistics for the VPN customer source and destination WAN connection provided for a given time.	5-217
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.	5-222
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.	5-226
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.	5-230
api_traffic_vpn_customer_wan_connection_topn_srcs	This query returns the traffic statistics for the top 100 sources for a particular WAN connection.	5-234
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.	5-238
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.	5-241
api_vpn_customer_wan_connection_get_config	This query returns the history statistics for the VPN customer for the given time period.	5-244

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.	5-251
api_vpn_enabled_customer_list_get_config	This query returns statistics for VPN traffic between the 100 selected WAN connections.	5-254

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 the RAMS:

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.

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.

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:
  - 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 RAMS 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 RAMS 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 all 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 RAMS or RAMS Traffic 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 $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;
$client = new RPC::XML::Client "http://$rexip: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 Route Analytics Management System XML RPC queries.

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/Sample

See Using Re-Entrant Queries on page 35 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}

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 "Expression Syntax" section in 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:
  - 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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}

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. 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 "Expression Syntax" section in 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/Sample

See Using Re-Entrant Queries on page 35 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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '2009.02.03.01.01.20.trunk-E Traffic
Explorer',
    'appliance_version' => '2009.02.03.01.01.20.trunk'
  },
  '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 "Expression Syntax" section in 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/Sample

See Using Re-Entrant Queries on page 35 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 "Expression Syntax" section in 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:

- link : MP link struct
- topology : topology struct
- sif : string (if has IGP)
- dif : string (if has IGP)
- metric : int (if has IGP)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    printf "usage: RouteAnalyzer.api_mp_links ip database\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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  'numReturnedEntries' => '150',
  'network_name' => 'LabRight',
  'report_time' => '20050725T23:40:42',
  'totalEntries' => '150',
```

```
  'result' => [
    {
      'link' => {
        'srcNode' => {
          'type' => 'RAMS',
          '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' => ...
    }
  ]
}
```

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/Sample

See Using Re-Entrant Queries on page 35 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.

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, the source address is an XML struct that contains the IPv6 address of router and mask length of 128.
- **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:
  - link : MP link struct
  - topology : topology struct
  - path_src : router struct
  - path_dst : prefix IP struct
  - path_cost : int
  - paths : array of the following:
    - —path : string
    - —cost : int
    - —num_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, error string, or if IGP. Not applicable for OSI networks
        - —dif : either MP IP struct, error string, or if IGP. Not applicable for OSI networks
        - —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
      - —metric : int (if IGP)
      - —protocol : string
      - —prefix : prefix struct

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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
  'appliance_version' => '2.5.0'
},
'numReturnedEntries' => '1',
'network_name' => 'lab',
'report_time' => '20061103T21:07:42',
'totalEntries' => '1',
'result' => [
  {
    'path_dst' => {
      'masklen' => '32',
      'ip_addr' => {
        'ip4_addr' => '24.0.0.12'
      }
    },
    'path_cost' => '-1',
    'path_src' => {
      'type' => 'ASBR',
      'ipaddr' => {
        'ip4_addr' => '24.0.0.2'
      }
    },
    'paths' => [
      {
        'cost' => '2',
        'path_hops' => [
          {
            'hop_src' => {
              'type' => 'ASBR',
              'ipaddr' => {
                'ip4_addr' => '24.0.0.2'
              }
            },
            'protocol' => 'OSPF',
            'hop_dst' => {
              'type' => 'LAN Pseudo-Node',
              'ipaddr' => {
                'ip4_addr' => '192.168.101.2'
              }
            },
            'interfaces' => [
```

```
                    {
                      'dif' => 'Not available',
                      'sif' => {
                        'ip4_addr' => '192.168.101.2'
                      }
                    }
            ],
                  'area/AS' => 'lab.OSPF/0.0.0.1',
                  'metric' => '1',
                  'prefix' => {
                    'masklen' => '32',
                    'ip_addr' => {
                      'ip4_addr' => '24.0.0.12'
                    }
                  }
                },
                ...
              ],
              'path' => 'Path 1',
              'num_hops' => '3'
            },
            ...
          ]
        }
      ]
    }
```

Sample Output (for IPv6) :

```
{
  'vinfo' => {
    'software_version' => 'Unversioned Traffic Explorer',
    'appliance_version' => 'unknown appliance version'
  },
  '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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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.
0000.0021.0000.01',
                '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.
0000.0021.0000.01',
                  '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_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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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.
0000.0021.0000.00',
        '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 "Expression Syntax" section in 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/Sample

See Using Re-Entrant Queries on page 35 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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_o
rigin',
                        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' => '2009.02.05.01.17-E Traffic
Explorer',
    'appliance_version' => 'unknown appliance version'
  },
  '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/Sample

See Using Re-Entrant Queries on page 35 for example and sample output details.

api_mp_routers

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

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 "Expression Syntax" section in 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:
  - 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '2009.02.03.01.01.20.trunk-E Traffic
Explorer',
    'appliance_version' => '2009.02.03.01.01.20.trunk'
  },
  '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 "Expression Syntax" section in 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 $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("24 Jun 2008 11:55:17 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_mp_routers_consolid
ated',
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' => '2008.06.24.11.42-E RAMS Traffic',
  'appliance_version' => '2008.06.18.02.32.11.trunk'
},
'numReturnedEntries' => '17',
'network_name' => 'PDlabDrex',
'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 "Expression Syntax" section in 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/Sample

See Using Re-Entrant Queries on page 35 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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '2009.02.26.05.01.48.7.5-E Traffic
Explorer',
    'appliance_version' => '2009.02.26.05.01.48.7.5'
  },
  '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' => 'rex212.packetdesign.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 "Expression Syntax" section in 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/Sample

See Using Re-Entrant Queries on page 35 for example and sample output details.

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. This query returns a subset of the results returned by the api_prefix_list query.

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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = 1058927123;

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_prefix_list_multi_o
rig',
           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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
```

```
'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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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, RAMS 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' => '5.5.0-R RAMS'
    'appliance_version' => '2.5.0'
  },
  '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 Route Analytics Management System 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.-E RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 RAMS or RAMS Traffic 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.

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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip: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_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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.0-E RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_pe_par
ticipation',
    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' => '5.5.0- RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_pe_lis
t',
    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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

```
push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_reacha
bility',
    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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";
```

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

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_reacha
bility_by_peer',
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' => '5.5.0-R RAMS HP OpenView Route
Analytics Management System',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip: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' => ' HP OpenView Route Analytics
Management System5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";
my $t1 = str2time("28 Jul 2004 08:25:51 PST");

push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_re
achability',
    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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $rexip = $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;
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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


push (@reqs,
RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_re
achability_by_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' => '5.5.0-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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 $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;
$client = new RPC::XML::Client "http://$rexip: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' => '5.5.11-R RAMS',
    'appliance_version' => '2.5.0'
  },
  '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 "Expression Syntax" section in 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/Sample

See explanation of re-entrant queries in Using Re-Entrant Queries on page 35.

5 VPN Customer Report Queries

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.

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.

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

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 $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 $t1 = str2time("20080922T12:30:00");
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer',
    RPC::XML::RPC_STRING($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' => '6.5.27-E RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 : integer (bps)
- min : integer (bps)
- max: integer (bps)
- ninetyfifthpctile (bps)

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
    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("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_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' => '6.5.27-E RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip:2002/RPC2";

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

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_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("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' => 'Unversioned RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip:2002/RPC2";

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

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_cust
omer_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' => 'Unversioned RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $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("20080922T12:30:00");
```

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_wan_connection',
    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' => '6.5.27-E RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $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("20080922T12:30:00");

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_wan_connection_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' => '6.5.27-E RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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, maxumum, 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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip:2002/RPC2";

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

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_wan_connection_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' => 'Unversioned RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 configuried 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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip:2002/RPC2";

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

```
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_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("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' => 'Unversioned RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $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("20080922T12:30:00");
push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_wan_connection_to_wan_connection',
```

Sample Output

```
{
  'vinfo' => {
    'software_version' => '6.5.27-E RAMS Traffic',
    'appliance_version' => '6.5.27'
  },
  '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 $rexip = $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';
$client = new RPC::XML::Client "http://$rexip:2002/RPC2";

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

push (@reqs,
RPC::XML::request->new('TrafficAnalyzer.api_traffic_vpn_custo
mer_wan_connection_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' => 'Unversioned RAMS Traffic',
    'appliance_version' => '6.5.27'
```

```
  },
  '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'
    }
  }
}
```