HP Route Analytics Management System

Software Version: 5.50

Developer's Guide

Manufacturing Part Number: BA129-90029
Document Release Date: August 2007
Software Release Date: August 2007

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.

Support

You can visit the HP software support web site at:

http://support.openview.hp.com/support.jsp

HP Software online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support 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 and HP Passport, go to:

http://support.openview.hp.com/new_access_levels.jsp

Contents

1 Introduction

You can create RAMS queries using an Application Programming Interface (API). RAMS 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 RAMS.
- Integrate RAMS with other tools you have that support XML RPC.

2 Configuring and Using Queries

This chapter describes how to configure and use RAMS queries. To use these queries from a program, it is necessary to link in the appropriate XML RPC library or package.

Note XML RPC is case sensitive.

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

Configuring RAMS to Accept Queries

Before you can use queries, you must configure RAMS 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. Select **Enable queries**.
3. 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.
4. Click **Update**.

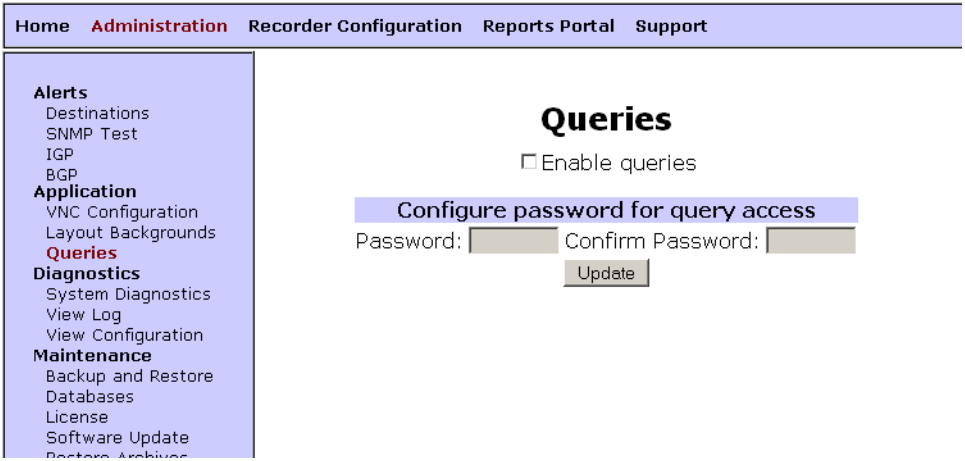

Figure 1 Queries Page

Using Queries

This appendix specifies the input parameters and results for the XML RPC calls listed in Table 1. 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 `api_list_a_route` queries apply to paths crossing both IGP and BGP domains. The calls with names beginning `api_vpn_` apply only to BGP/MPLS VPN protocol domains. The remainder apply only to IGP protocol domains.

> **Note** 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 appendix. The Dumper function is included in the standard Perl package called Data.

Table 1 Query Calls and Descriptions

Query	Description	Page
api_del_watch_list	Delete the watch list for the specified alert.	32
api_get_alert_destination_info	Return information about the destination to which the specified alert is sent.	34
api_get_watch_list	Return the watch list for the specified alert.	37
api_link_list	Return a list of links for the specified network.	40
api_list_a_route	Return the total metric of a path (route) and the list of links.	43
api_list_a_route_ECMP	Return the total metric of a path (route) and the list of links, including all ECMP paths.	47
api_mp_close_handle	This query takes a previously generated report handle and closes the report, freeing the memory and resources it uses	51
api_mp_events	Lists all multi-protocol network events between two different times.	52
api_mp_events_handle	Lists all multi-protocol network events between two different times in chunks of a user-specified size.	56

Query	Description	Page
api_mp_links	Lists links meeting filter criteria in a multi-protocol network.	64
api_mp_list_handle	Returns a user-specified number of entries starting at a user-specified point in the report	68
api_mp_routes	Lists prefix routes meeting filter criteria in a multi-protocol network.	85
api_mp_routes_handle	List prefix routes meeting filter criteria in a multi-protocol network in chunks of a user-specified size.	89
api_mp_routers	Lists routers meeting filter criteria in a multi-protocol network.	90
api_prefix_events	Return a list of events for the specified prefix.	94
api_prefix_list_filtered	Return a list of prefixes for the specified router.	101
api_prefix_list_multi_orig	Return a list of prefixes that are originated by more than one router for the specified network.	104
api_prefix_list_same	Lists all prefixes advertised by multiple routers at the same time.	108
api_resource_status	Lists the current status of the memory, disk, and swap space on the appliance.	111
api_router_events	Return a list of events associated with the specified router.	115
api_router_list	Return a list of routers for the specified network.	119
api_router_summarizable	Lists routers advertising multiple summarizable prefixes.	122
api_system_health	Lists the health of all the Route Explorer and Traffic Explorer systems in the network, including the recording and writing status of each configured recording process and its databases.	125
api_vpn_cust_rt_list	Lists all customer names in VPN to RT (Route Target) mappings.	129

Query	Description	Page
api_vpn_customer_pe_participation	Return statistics of participating PEs for each VPN customer.	132
api_vpn_customer_pe_list	Return the list of participating PEs for the specified VPN customer.	136
api_vpn_customer_reachability	Return reachability statistics for each VPN customer.	140
api_vpn_customer_reachability_by_peer	Return reachability statistics at each PE for the specified VPN customer.	144
api_vpn_route_target_pe_participation	Return statistics of participating PEs for each route target in the specified network.	149
api_vpn_route_target_pe_list	Return the list of participating PE routers and their VPN state for the specified route target.	153
api_vpn_route_target_reachability	Return reachability statistics for each route target in the specified network.	157
api_vpn_route_target_reachability_by_peer	Return reachability statistics at each PE for the specified route target.	162
api_vpn_routes	Return the list of VPN routes for the specified network.	167
api_vpn_routes_handle	Lists all prefixes advertised in the network.	171

Understanding Fault Codes

The fault codes are listed in Table 2 and are used in the RAMS:

Table 2 XML RPC Fault Codes

Code	Description
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.
213	Watch list does not exist.
214	Invalid IP address struct in request.
215	Source router does not exist.
216	Path is of length 0.
219	Invalid trap name.
220	Unable to load routing events from database.
221	Unable to process system resource information.
222	Could not connect to one or more database(s)

Alert Values

Table 3 lists the alert values and their corresponding alert names.

Table 3 Alert Values and Names

Alert Value	Alert Name
AdjLost	Adjacency Lost
AdjEst	Adjacency Established
LnkFlap	Adjacency Flap
RtChange	Prefix Change
RtOrigChange	Prefix Origination Change
PrefixFlap	Prefix Flap
RtFlap	Route Change
Event	Routing Event
EChurn	Excess Churn
PeerChange	Peer Change
BgpPrefixDrought	Prefix Drought
BgpPrefixFlood	Prefix Flood
BgpRouteFlap	Route Flap
BgpLostRedund	Lost Redundancy
BgpLostPeer	Lost Peer
BgpEstPeer	Established Peer
BgpAcqRedund	Acquired Redundancy
BgpASPathLonger	AP Path Longer
BgpDownToOnePath	Down to one path
BgpDownToZeroPaths	Down To Zero Paths
RtrReachability	Customer Reachability by PE
CustReachability	Customer Reachability
CustPrivacy	Customer PE Participation
RtrIntrusion	New Customer PE

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 us)
- 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
- 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)
  - type: string
  - sysid: string (if protocol is ISIS)
  - protoType: 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 us * 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

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_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 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. The only 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 example below, 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:

```
#!/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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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 RAMS XML RPC queries.

api_del_watch_list

RPC Call: Route Analyzer.api_del_watch_list {password} {alert}

This query deletes the watch list that was previously set up for an alert. A watch list limits alerts to events matching specific criteria, which could be a combination of (prefix and mask) or (source router and destination router) depending on the specific alert. By deleting the watch list, the limits on the alert will be removed. The query returns a 1 on success, 0 otherwise.

Input Parameters

- **password** – The password configured for queries.
- **alert** – The alert for which the watch list will be deleted. Currently, watch lists can be deleted for the following alerts: AdjLost, AdjEst, RtFlap, LnkFlap, RtOrigChange, RtChange, PeerChange, BgpRouteFlap, BgpLostRedund. Detailed information about alerts can be found in the "Alerts" chapter in the *Users Guide*.

Structure of Output

- boolean

Example

```
#!/usr/bin/perl

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

my $rexip = $ARGV[0];

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

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

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---
'0'
```

api_get_alert_destination_info

RPC Call: RouteAnalyzer.api_get_alert_destination_info {password} {alert}

This query returns information about the alert destination that was configured using the web administrative interface as described in the “VPN Configuration and Reports” chapter in the *User’s Guide*. This includes the alert destination, the SNMP Trap Community, SNMP Trap Destination, the threshold, and the time scale.

Input Parameters

- **password** – The password configured for queries.
- **alert** – The alert for which destination information is requested. Currently, the following alerts are implemented: AdjLost, AdjEst, RtFlap, LnkFlap, RtOrigChange, RtChange, Event, EChurn, PeerChange, PrefixFlap, BgpPrefixDrought, BgpPrefixFlood, BgpRouteFlap, BgpLostRedund, BgpASPathLonger, BgpDownToOnePath, BgpDownToZeroPaths, BgpAcquiredRedund, RtrReachability, CustReachability, CustPrivacy, RtrIntrusion, BgpPeerLost. Detailed information about alerts can be found in the "Alerts" chapter in the *Users Guide*.

Structure of Output

- vinfo : version struct
- alert_dest:
  - snmp_trap_community: string
  - time_scale : int
  - threshold : int (percentage)
  - alert_destination : string
  - snmp_trap_destination : string

Example

```
#!/usr/bin/perl

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

my $rexip = $ARGV[0];

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

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

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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'alert_dest' => {
      'snmp_trap_community' => 'public',
      'time_scale' => '60',
      'threshold' => '50',
      'alert_destination' => 'SNMP Trap',
      'snmp_trap_destination' => '192.168.0.123'
   }
}
```

api_get_watch_list

RPC Call: RouteAnalyzer.api_get_watch_list {password} {alert}

This query returns the watch list that have been set up for the specified alert. Watch lists limit alerts to events matching specific criteria, which could be a combination of (prefix and mask) or (source router and destination router) depending on the specific alert. The example output shows a watch list that has been set up for the RtFlap alert. The output shows the source router, the destination router, and an “op” value. The “op” value can be: “and,” “or”, or “none.”

Input Parameters

- **password** – The password configured for queries.
- **alert** – The alert for which the list of set watch lists is requested. Currently, watch lists can be obtained from the following alerts: AdjLost, AdjEst, RtFlap, LnkFlap, RtOrigChange, RtChange, PeerChange, BgpRouteFlap, BgpLostRedund. Detailed information about alerts can be found in the "Alerts" chapter in the *Users Guide*.

Structure of Output

- vinfo : version struct
- wlists: array of the following:
  - src : IP struct
  - dst : IP struct
  - op : string

Example

```
#!/usr/bin/perl

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

my $rexip = $ARGV[0];
use strict;
use RPC::XML::Client;
use Date::Parse;
use Data::Dumper; $Data::Dumper::Terse = 1; $Data::Dumper::
   Indent = 1;
my $client;
my $req;
my @reqs;
my $password = 'admin';
my $alert = "RtFlap";
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'wlists' => [
      {
         'src' => {
         'ip4_addr' => '192.168.0.123'
         },
         'dst' => {
         'ip4_addr' => '192.168.0.40'
         },
         'op' => 'and'
      }
   ]
}
```

api_link_list

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

This query returns a list of links for the specified network. The information includes, for each link, a description of the source node, destination node, and the interface between the two.

> **Note** This query is deprecated and may be removed in a future release. New applications should use api_mp_links instead.

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
- links : array of link structs

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_link_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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = str2time("24 Jul 2003 11:53:33 PST");

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_link_list',
      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 = $res->value;
   print (STDERR join "\n", "---XMLRPC RESULT value1---",
      Dumper($value1) );
}
```

Sample Output

```
---XMLRPC RESULT value1---
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'UCBJul03a',
   'report_time' => '20051027T18:46:09',
   'links' => [
      {
         'source' => {'nodeType' => 'Internal',
            'ip_addr' => {'ip4_addr' => '169.229.128.130'},
            'nodeState' => 'UP',
            'nodeProto' => 'OSPF',
            'name' => '',
            'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
            'maskLen' => '32',
            'systemID' => '169.229.128.130'
         },
         'interfaces' => [
            {
               'source' => {'ip4_addr' => '169.229.128.130'},
               'bw' => '0',
               'destination' => {'ip4_addr' => '169.229.128.129'},
               'metric' => '1000',
               'delay' => '10000',
               'state' => '1'
            }
         ],
      'destination' => {
            'nodeType' => 'PseudoNode',
            'ip_addr' => {'ip4_addr' => '169.229.128.129'},
            'nodeState' => 'UP',
            'nodeProto' => 'OSPF',
            'name' => '',
            'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
            'maskLen' => '29',
            'systemID' => '169.229.128.129 DR'
         }
      },...
   ]
}
```

api_list_a_route

RPC Call: Route Analyzer.api_list_a_route {password} {database name} {source address} {dest prefix} {time}

This query returns the total metric (if calculable) and the list of links in the path from the source to the destination at the requested time. Only one path is listed even if multiple equal-cost paths exist. The list of links is in order.

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.
- **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`.
- **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
- route :
  - route_cost : int
  - time : ISO 8601 UTC time
  - links : array of link structs

Example

```
#!/usr/bin/perl

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 = 'admin';

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

my $t1 = 1099528204;    #seconds since 1970-01-01 00:00:00 UTC

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_list_a_route',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      ##### source #####
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "192.168.99.99"),
         masklen => 32),
      ##### destination #####
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "10.23.113.0"),
         masklen => 27),
      RPC::XML::datetime_iso8601->new(time2iso8601($t1))
   )
);

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

Sample Output

```
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'tester',
   'report_time' => '20041104T00:31:49',
   'route' => {
      'route_cost' => '5',
      'time' => '20041104T00:30:04',
      'links' => [
           {'source' => {'nodeType' => 'AreaBR_ASBR',
              'ip_addr' => {'ip4_addr' => '192.168.99.99'},
              'name' => '',
              'nodeArea' => 'tester.OSPF/Backbone',
              'systemID' => '192.168.99.99',
              'nodeState' => 'UP',
              'nodeProto' => 'Unknown',
              'maskLen' => '32'
           },
           'interfaces' => [
              {
                 'source' => {'ip4_addr' => '192.168.116.11'},
                 'bw' => '0',
                 'destination' => {'ip4_addr' => '192.168.116.17'},
                 'metric' => '1',
                 'delay' => '10000',
                 'state' => '1'
              }
           ],
           'destination' => {'nodeType' => 'PseudoNode',
              'ip_addr' => {'ip4_addr' => '192.168.116.17'},
              'name' => '',
              'nodeArea' => 'tester.OSPF/Backbone',
              'systemID' => '192.168.116.17 DR',
              'nodeState' => 'UP',
              'nodeProto' => 'Unknown',
              'maskLen' => '24'
           }
         },...
      ]
   }
}
```

api_list_a_route_ECMP

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

This query returns the total metric (if calculable) and the collection of links comprising one or more equal-cost paths from the source to the destination at the requested time. The list of links is the result of a breadth-first search of the tree of path segments. Consequently, the links may not be listed in order for any of the paths and links that are present on multiple paths may be replicated in the list.

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.
- **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`.
- **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
- router:

- route_cost : int
- time : ISO 8601 UTC time
- links : array of link structs

Example

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

my $t1 = str2time("3 Nov 2004 16:30:04 PST");

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_list_a_route_ECMP',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      ##### source #####
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "192.168.99.99"),
         masklen => 32),
      ##### destination #####
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "10.23.113.0"),
         masklen => 27),
      RPC::XML::datetime_iso8601->new(time2iso8601($t1))
   )
);

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

Sample Output

```
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'tester',
   'report_time' => '20041105T21:24:16',
   'route' => {
      'route_cost' => '8',
      'time' => '20041104T00:30:04',
      'links' => [
           {'source' => {'nodeType' => 'AreaBR_ASBR',
              'ip_addr' => {'ip4_addr' => '192.168.99.99'},
              'name' => '',
              'nodeArea' => 'tester.OSPF/Backbone',
              'systemID' => '192.168.99.99',
              'nodeState' => 'UP',
              'nodeProto' => 'Unknown',
              'maskLen' => '32'
           },
           'interfaces' => [
              {
                 'source' => {'ip4_addr' => '192.168.99.99'},
                 'bw' => '0',
                 'destination' => {'ip4_addr' => '192.168.107.11'},
                 'metric' => '2',
                 'delay' => '10000',
                 'state' => '1'
              }
           ],
           'destination' => {'nodeType' => 'PseudoNode',
              'ip_addr' => {'ip4_addr' => '192.168.107.11'},
              'name' => '',
              'nodeArea' => 'tester.OSPF/Backbone',
              'systemID' => '192.168.107.11 DR',
              'nodeState' => 'UP',
              'nodeProto' => 'Unknown',
              'maskLen' => '24'
           }
         },...
      ]
   }
}
```

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 27 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.

> **Note** 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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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} {max entries}

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.

> **Note** 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

See input parameters for api_mp_events on page 52

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 27 for example and sample details.

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.

> **Note** 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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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} {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.

> **Note** 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

See input parameters for api_mp_osi_routes on page 57for input parameters

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 27 for example and sample 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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'numReturnedEntries' => '150',
   'network_name' => 'LabRight',
   'report_time' => '20050725T23:40:42',
   'totalEntries' => '150',
   'result' => [
      {
         'link' => {
            'srcNode' => {
               'type' => 'Route Explorer',
               'ipaddr' => '192.168.122.90'
               },
               'dstNode' => {
                  'type' => 'IBGP Peer',
                  'ipaddr' => '192.168.100.100'
               },
               'state' => {
                  'down' => 'false'
               }
            },
            'topology' => {
               'fullName' => 'LabRight.ConfedsTest.ConfedTestTop.BGP
                  /AS65510',
               'protocol' => 'BGP'
            }
         },
         {
         'link' => {
            'srcNode' => {'type' => ..., 'ipaddr' => ...},
            'dstNode' => {'type' => ..., 'ipaddr' => ...},
            'state' => {'down' => ...}, //end of link
            'dif' => ...,
            'sif' => ...,
            'topology' => {'fullName' => ..., 'protocol' => ...}
         }, //end of topology
         'metric' => ...
      }
   ]
}
```

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.

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 calcuable) 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.
- **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.`
- **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 us * 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

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::Indemy
$password = 'admin';nt = 1;
my $client; my $req; my @reqs;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = 1099528204; #seconds since 1970-01-01 00:00:00 UTC

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_list_a_route',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
   ##### source #####
   new RPC::XML::struct(ip_addr =>
      new RPC::XML::struct(ip4_addr => "24.0.0.2"),
      masklen => 32),
   ##### destination #####
   new RPC::XML::struct(ip_addr =>
      new RPC::XML::struct(ip4_addr => "24.0.0.12"),
      masklen => 27),
   RPC::XML::datetime_iso8601->new(time2iso8601($t1))
   )
);

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

Sample Output

```
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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'
    },...
  ]
 }
]
}
```

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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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_prefixes_multi_origin

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

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.
- **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 (IP prefixes or Prefix Neighbors/ES Neighbor) : 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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_
      origin',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
      RPC::XML::RPC_INT($thresh),
      RPC::XML::RPC_INT($num)
   )
);
```

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

Sample Output

```
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'numReturnedEntries' => '6',
   'network_name' => 'PDIHari',
   'report_time' => '20070206T18:33:03',
   'totalEntries' => '6',
   'result' => [
      {
         'nsap' => '47.0010.0001',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         'router' => {
            'sysid' => '47.0024.0000.0001.0000.00,27.0001.0000.0001.
               0000.00',
            'name' => 'Router1',
            'type' => 'L1L2 Router',
            'protoType' => 'OSI'
         }
      },
      {
         'nsap' => '47.0010.0001',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         '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'
         }
      },
      {
         'nsap' => '47.0011.0001',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         'router' => {
            'sysid' => '47.0024.0000.0001.0000.00,27.0001.0000.
               0001.0000.00',
            'name' => 'Router1',
            'type' => 'L1L2 Router',
```

```
            'protoType' => 'OSI'
         }
      },
      {
         'nsap' => '47.0011.0001',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         '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'
         }
      },
      {
         'nsap' => '00',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         'router' => {
            'sysid' => '47.0024.0000.0001.0000.00,27.0001.0000.
              0001.0000.00',
            'name' => 'Router1',
            'type' => 'L1L2 Router',
            'protoType' => 'OSI'
         }
      },
      {
         'nsap' => '00',
         'area' => 'PDIHari.ISIS/Level2',
         'type' => 'Prefix Neighbor',
         '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'
         }
      }
   ]
}
```

api_mp_prefixes_multi_origin_handle

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

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

See api_mp_prefixes_multi_origin for input parameters.

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 (IP prefixes or Prefix Neighbors/ES Neighbor) : string
  - prefix_area : string
  - prefix_type : string
  - router : MP router struct

Example/Sample

See Using Re-Entrant Queries on page 27 for example and sample output 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.

> **Note** 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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'numReturnedEntries' => '150',
   'network_name' => 'baklab701',
   'report_time' => '20051107T23:07:45',
   'totalEntries' => '680',
   'result' => [
      {
         'topology' => {
            'fullName' => 'AllStaticRoutes.Static',
            'protocol' => 'Static'
         },
         'attributes' => {
            'nextHops' => [
               {
                  'nextHop' => '192.168.101.101/8'
               }
            ]
         },
         'prefix' => '0.0.0.0/0',
         'router' => {
            'type' => 'Static',
            'ipaddr' => '192.168.133.34'
         },
         'state' => {
            'inBaseline' => 'false',
            'down' => 'false'
         }
      },...
   ]
}
```

api_mp_routes_handle

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

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..

> **Note** 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

See api_mp_routes on page 85 for input parameters.

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 27 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 *Users 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

— 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'numReturnedEntries' => '100',
   'network_name' => 'pd353',
   'report_time' => '20051027T23:33:57',
   'totalEntries' => '100',
   'result' => [
      {
         'topology' => {
            'fullName' => 'pd353.Left.BGP/AS65522',
            'protocol' => 'BGP'
         },
         'numPrefixes' => '0',
         'router' => {
            'type' => 'Route Explorer',
            'ipaddr' => '192.168.0.49'
         },
         'state' => {
            'down' => 'false'
         }
      },
      {
         'topology' => {
            'fullName' => 'pd353.Left.ISIS/Level2',
            'protocol' => 'ISIS'
         },
         'numPrefixes' => '3',
         'router' => {
            'name' => 'labnet-gw',
            'type' => 'AreaBR',
            'ipaddr' => '192.168.104.254'
         },
         'state' => {
            'down' => 'false'
         }
      },...
   ]
}
```

api_prefix_events

RPC Call: RouteAnalyzer.api_prefix_events {password} {database name} {prefix}

This query returns a list of events for the specified prefix. Each event listing includes information about the prefix and the router that advertised that prefix.

> **Note** The query returns the list of events for the specified prefix for the entire time interval covered by the requested database. This could result in a large number of entries returned in the result. To limit the number of entries, you can use the api_mp_events query instead by specifying the prefix as the filter.

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.
- **prefix** – An XML struct that contains any prefix consisting of an IPv4 address, such as `192.168.123.125`, and a mask length, such as `27`.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- events: array of the following:
  - prefix_event:
    - eventType: string
    - time: ISO 8601 UTC time
    - usec: int

— prefix: prefix struct

— router: router struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_prefix_events 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_prefix_events',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "169.229.147.0"),
         masklen => 25)
   )
);

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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'UCBJul03a',
   'report_time' => '20051027T18:42:29',
   'events' => [
      {
         'prefix_event' => {
            'eventType' => 'advertised',
            'time' => '20030722T22:10:56',
            'usec' => '521406',
            'prefix' => {
               'masklen' => '25',
               'ip_addr' => {'ip4_addr' => '169.229.147.0'}
            },
            'router' => {
               'nodeType' => 'AreaBR_ASBR',
               'ip_addr' => {'ip4_addr' => '128.32.1.209'},
               'nodeState' => 'UP',
               'nodeProto' => 'OSPF',
               'name' => '',
               'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
               'maskLen' => '32',
               'systemID' => '128.32.1.209'
            }
         }
      },...
   ]
}
```

api_prefix_list

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

This query returns the list of all prefixes advertised in the database at a particular time. The example program below sends a request for the advertised prefixes at midnight on December 17, 2002 and displays the prefix information including the prefix IP address, type, area, and advertising routers.

> **Note** This query is deprecated and may be removed in a future release. New applications should use api_mp_routes instead.

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:
  - prefix_type: string
  - prefix_area: string
  - prefix: prefix struct
  - routers: array of router 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(str2time("17 Dec 2002 00:00:00 PST"));
my $request = RPC::XML::request->new(
   'RouteAnalyzer.api_prefix_list',
   RPC::XML::RPC_STRING( 'packet' ),  //password
   RPC::XML::RPC_STRING( 'CorpNet' ), //database name
   RPC::XML::datetime_iso8601->new($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' => 'Build 4.0.90 HP RAMS'
      'appliance_version' => '0.5.24',
   },
   'report_time' => '20030303T20:42:01',
   'network_name' => 'CorpNet',
   'prefixes' => [
      {
         'prefix_type' => 'Internal',
         'prefix_area' => '00000001/ospf',
         'prefix'=>{'ip_addr'=>{'ip4_addr'=>'192.168.240.1'},'maskl
            en'=>'32'},
         'routers' => [
            {
               'nodeProto' => 'ospf',
               'ip_addr' => { 'ip4_addr' => '192.168.104.2' },
               'nodeType' => 'ASBR',
               'name' => '',
               'systemID' => '192168104002:00'
            }
            {'nodeProto'...'ip_addr'...'nodeType'...'name'...
               'systemID'...},
            {'nodeProto'...'ip_addr'...'nodeType'...'name'...
               'systemID'...}
         ] //end routers
      },
      {'prefix_type'...'prefix_area'...'prefix'...'routers'=>
         [...] },
      {'prefix_type'...'prefix_area'...'prefix'...'routers'=>
         [...] }
   ] //end prefixes
}
```

api_prefix_list_filtered

RPC Call: RouteAnalyzer.api_prefix_list_filtered {password} {database name} {router} {time}

This query returns a list of prefixes originated by the specified router at the specified time.

> **Note** This query is deprecated and may be removed in a future release. New applications should use api_mp_routes instead.

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.
- **router** – 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.
- **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_filtered 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_prefix_list_filtered',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      RPC::XML::datetime_iso8601->new(time2iso8601($t1)),
      new RPC::XML::struct (ip4_addr => "192.168.120.120")
   )
);

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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'Lab',
   'report_time' => '20051031T22:58:04',
   'prefixes' => [
      {
         'routers' => [
            {
               'nodeType' => 'ASBR',
               'ip_addr' => {'ip4_addr' => '192.168.120.120'},
               'nodeState' => 'DOWN',
               'nodeProto' => 'Static',
               'name' => 'Router16',
               'nodeArea' => 'Lab.EIGRP/AS1',
               'maskLen' => '32',
               'systemID' => '192.168.120.120'
            },
            {
               'nodeType' => 'Internal',
               'ip_addr' => {'ip4_addr' => '192.168.220.20'},
               'nodeState' => 'DOWN',
               'nodeProto' => 'Static',
               'name' => 'Router20',
               'nodeArea' => 'Lab.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_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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = 1058927123;

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

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

Sample Output

```
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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_prefix_list_same

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

This query returns all routers advertising the same prefix or a more specific prefix for all prefixes advertised in the network by the specified database. Prefixes that are internal (native to the IGP) and those that are external (imported from another routing protocol) are compared 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 zonetime 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:
  - prefix_type: string
  - prefix_area: string
  - prefix: prefix struct
  - routers: array of router 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(str2time("17 Dec 2002 00:00:00 PST"));
my $request = RPC::XML::request->new(
   'RouteAnalyzer.api_prefix_list_same',
   RPC::XML::RPC_STRING( 'admin' ),   //password
   RPC::XML::RPC_STRING( 'CorpNet' ),  //database name
   RPC::XML::datetime_iso8601->new($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' => 'Build 4.0.90 HP RAMS'
      'appliance_version' => '0.5.24',
   },
   'report_time' => '20030303T20:52:20',
   'network_name' => 'CorpNet',
   'prefixes' => [
      {
         'prefix_type' => 'Area-Ext',
         'prefix_area' => '00000001/ospf',
         'prefix' => {
            'ip_addr' => {'ip4_addr' => '192.168.200.200'},
            'masklen' => '24'
         },
         'routers' => [
            {
               'nodeProto' => 'ospf',
               'ip_addr' => {'ip4_addr' => '192.168.201.201'},
               'nodeType' => 'ASBR',
               'name' => '',
               'systemID' => '091001011001:00'
            },
            {'nodeProto'...'ip_addr'...'nodeType'...'name'
               ...'systemID' ...}
            {'nodeProto'...'ip_addr'...'nodeType'...'name'
               ...'systemID' ...}
         ] // end routers
      }, // end first prefix
      {'prefix_type'...'prefix_area'...'prefix'.... 'routers'
         => [...]},
      {'prefix_type'...'prefix_area'...'prefix'.... 'routers'
         => [...]}
   ] // end prefixes
}
```

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
- resouces:
  - 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;
my $password = 'admin';

$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'resources' => {
      'memory' => {
         'pct' => '71.70806685821979',
         'free' => '293032',
         'used' => '742712',
         'total' => '1035744'
      },
      'disk' => {
         'pct' => '79.65773029037497',
         'free' => '5195956',
         'used' => '27265044',
         'total' => '34227744'
      },
      'cpu' => {
         'system' => '0',
         'other' => '0',
         'user' => '0',
         'idle' => '100'
      },
      'swap' => {
         'pct' => '2.690779962482124',
         'free' => '2985840',
         'used' => '82564',
         'total' => '3068404'
      }
   }
}
```

api_router_events

RPC Call: RouteAnalyzer.api_router_events {password} {database name} {router}

This query returns a list of events associated with the specified router.

> **Note** This query is deprecated and may be removed in a future release. New applications should use api_mp_events instead.

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.
- **router** – 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.

Structure of Output

- vinfo: version struct
- network_name: string
- report_time: ISO 8601 UTC time
- events: array of the following:
  - router event:
    - eventType: string
    - time: ISO 8601 UTC time
    - usec: int
    - router: router struct

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_router_events 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_router_events',
      RPC::XML::RPC_STRING($password),
      RPC::XML::RPC_STRING($database),
      new RPC::XML::struct(ip_addr =>
         new RPC::XML::struct(ip4_addr => "128.32.1.211"))
   )
);

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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'network_name' => 'UCBJul03a',
   'report_time' => '20051027T19:01:13',
   'events' => [
      {
         'router_event' => {
            'eventType' => 'advertised',
            'time' => '20030722T22:10:56',
            'usec' => '581591',
            'router' => {
               'nodeType' => 'ASBR',
               'ip_addr' => {'ip4_addr' => '128.32.1.211'},
               'nodeState' => 'DOWN',
               'nodeProto' => 'OSPF',
               'name' => '',
               'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
               'maskLen' => '32',
               'systemID' => '128.32.1.211'
            }
         }
      },
      {
         'router_event' => {
            'eventType' => 'advertised',
            'time' => '20030722T22:10:52',
            'usec' => '851696',
            'router' => {
               'nodeType' => 'ASBR',
               'ip_addr' => {
                  'ip4_addr' => '128.32.1.211'
               },
               'nodeState' => 'DOWN',
               'nodeProto' => 'OSPF',
               'name' => '',
               'nodeArea' => 'UCBJul03a.OSPF/169.229.128.168',
               'maskLen' => '32',
               'systemID' => '128.32.1.211'
            }
```

```
            }
        },...
    ]
}
```

api_router_list

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

This query returns a list of routers for the specified network.

> **Note** This query is deprecated and may be removed in a future release. New applications should use api_mp_routers instead.

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
- routers: array of router structs

Example

```
#!/usr/bin/perl

if(!defined($ARGV[0]) || !defined($ARGV[1])) {
   printf "usage: RouteAnalyzer.api_router_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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = str2time("24 Jul 2003 11:00:00 PST");

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_router_list',
      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 = $res->value;
   print (STDERR join "\n", "---XMLRPC RESULT value1---",
      Dumper($value1) );
}
```

Sample Output

```
---XMLRPC RESULT value1---
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'routers' => [
      {
         'nodeType' => 'Internal',
         'ip_addr' => {'ip4_addr' => '169.229.128.130'},
         'nodeState' => 'UP',
         'nodeProto' => 'OSPF',
         'name' => '',
         'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
         'maskLen' => '32',
         'systemID' => '169.229.128.130'
      },
      {
         'nodeType' => 'AreaBR_ASBR',
         'ip_addr' => {'ip4_addr' => '128.32.1.209'},
         'nodeState' => 'UP',
         'nodeProto' => 'OSPF',
         'name' => '',
         'nodeArea' => 'UCBJul03a.OSPF/169.229.128.128',
         'maskLen' => '32',
         'systemID' => '128.32.1.209'
      },
      {
         'nodeType' => 'Internal',
         'ip_addr' => {'ip4_addr' => '169.229.2.66'},
         'nodeState' => 'UP',
         'nodeProto' => 'OSPF',
         'name' => '',
         'nodeArea' => 'UCBJul03a.OSPF/169.229.128.168',
         'maskLen' => '32',
         'systemID' => '169.229.2.66'
      },
   ],
   'network_name' => 'UCBJul03a',
   'report_time' => '20051028T00:00:05'
}
```

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. 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' => 'Build 4.0.90 HP RAMS'
      'appliance_version' => '0.5.24',
   },
   '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 Router Explorer and Traffic Explorer systems in the network, including the recording and writing status of each configured recording process and its databases. Non-master units can only look at their local unit, while master units can look at the status of each of their clients.

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

Input Parameters

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

Structure of Output

- vinfo: version struct
- units: array of the following:
  - reachable : boolean
  - ipaddr : string
  - processes : array of the following:
    - globaldbname : string
    - running : boolean
    - process : string
    - dbs : array of the following:
    - dbname : string
    - messages : array of the following:
    - msg : string
    - last_write_time: ISO 8601 UTC time

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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   'units' => [
      {
         'reachable' => '0',
         'processes' => [],
         'ipaddr' => '192.168.3.44'
      },
      {
         'reachable' => '1',
         'processes' => [
            {
               'label' => 'Traffic1',
               'running' => '0',
               'dbs' => [],
               'process' => 'Flow Recorder'
            }
         ],
         '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_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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_pe_
      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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";
my $t1 = str2time("30 Aug 2005 00:26:30 PDT");

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

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

Sample Output

```
---XMLRPC RESULT value1---
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";
my $t1 = str2time("28 Feb 2005 15:50:22 PST");

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

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

Sample Output

```
---XMLRPC RESULT value1---
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_vpn_customer_
      reachability_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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

my $t1 = str2time("28 Jul 2004 08:25:51 PST");

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

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

Sample Output

```
---XMLRPC RESULT value1---
{
   'vinfo' => {
      'software_version' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$rexip:2000/RPC2";

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

push (@reqs,
   RPC::XML::request->new('RouteAnalyzer.api_vpn_route_target_
      reachability_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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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 *Users 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;
my $password = 'admin';
$client = new RPC::XML::Client "http://$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' => 'Build 4.0.90 HP RAMS',
      'appliance_version' => '0.5.24'
   },
   '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} {max_entries}

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

Input Parameters

See api_vpn_routes on page 167 for input parameters.

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 27.

Index