Analytics and Location Engine 1.2

User and API Guide

ARUBA networks

Copyright Information

Open Source Code

Certain Aruba products include Open Source software code developed by third parties, including software code subject to the GNU General Public License (GPL), GNU Lesser General Public License (LGPL), or other Open Source Licenses. Includes software fro Litech Systems Design. The IF-MAP client library copyright 2011 Infoblox, Inc. All rights reserved.This product includes software developed by Lars Fenneberg et al. The Open Source code used can be found at this site

http://www.arubanetworks.com/open_source

Legal Notice

The use of Aruba Networks, Inc. switching platforms and software, by all individuals or corporations, to terminate other vendors' VPN client devices constitutes complete acceptance of liability by that individual or corporation for this action and indemnifies, in full, Aruba Networks, Inc. from any and all legal actions that might be taken against it with respect to infringement of copyright on behalf of those vendors.

Warranty

This hardware product is protected by the standard Aruba warranty of one year parts/labor. For more information, refer to the ARUBACARE SERVICE AND SUPPORT TERMS AND CONDITIONS.

The wireless network has a wealth of information about unassociated and associated devices. The Analytics and Location Engine (ALE) is designed to gather that information from the network, process it and share it through a simple and standard API. ALE includes a location engine that calculates associated and unassociated device location every 30 seconds by default. ALE needs the AP placement data to be able to calculate location for these devices. This map data must be imported from VisualRF.

For every device on the network, ALE provides the following information through the Northbound API:

- Client Username,
- IP address
- MAC address
- device type
- Application firewall data, showing the destinations and applications used by associated devices.
- current location
- historical location

Personal identifying information (PII) can be filtered out of the data, allowing you to view standard or anonymized data with or without MAC addresses, client user names, and IP addresses.

This guide describes how to install and configure ALE and includes information about ALE polling APIs, push and subscribe APIs, security (SSL bridge), and troubleshooting tips.

Conventions

The following conventions are used throughout this manual to emphasize important concepts:

Table 1: *Typographical Conventions*

Type Style	Description
Italics	This style is used to emphasize important terms and to mark the titles of books.
`System items`	This fixed-width font depicts the following: • Sample screen output • System prompts • Filenames, software devices, and specific commands when mentioned in the text
`Commands`	In the command examples, this bold font depicts text that you must type exactly as shown.
<*Arguments*>	In the command examples, italicized text within angle brackets represents items that you should replace with information appropriate to your specific situation. For example: # **`send`** <*text message*> In this example, you would type “send” at the system prompt exactly as shown, followed by the text of the message you wish to send. Do not type the angle brackets.
`[Optional]`	Command examples enclosed in brackets are optional. Do not type the brackets.
`{Item A | Item B}`	In the command examples, items within curled braces and separated by a vertical bar represent the available choices. Enter only one choice. Do not type the braces or bars.

The following informational icons are used throughout this guide:

Indicates helpful suggestions, pertinent information, and important things to remember.

Indicates a risk of damage to your hardware or loss of data.

Indicates a risk of personal injury or death.

Related Documents

The following documents are part of the complete documentation set for the ALE:

- *Analytics and Location Engine 1.2 Release Notes*
- *Aruba Instant 6.3.1.1 - 4.0 User Guide*
- *ArubaOS 6.x Quick Start Guide*
- *ArubaOS 6.4 User Guide*
- *ArubaOS 6.4 Command-Line Reference Guide*
- *ArubaOS 6.4 Release Notes*
- *Airwave 7.7 User and API Guide*

Contacting Aruba Networks

Table 2: *Contact Information*

Website Support	
Main Site	http://www.arubanetworks.com
Support Site	https://support.arubanetworks.com
Airheads Social Forums and Knowledge Base	http://community.arubanetworks.com
North American Telephone	1-800-943-4526 (Toll Free) 1-408-754-1200
International Telephone	http://www.arubanetworks.com/support-services/aruba-support-program/contact-support/
Support Email Addresses	
Americas and APAC	support@arubanetworks.com
EMEA	emea_support@arubanetworks.com
Wireless Security Incident Response Team (WSIRT) .	wsirt@arubanetworks.com

This topic discusses ALE setup and installation instructions. Information includes:

ALE is designed to gather client information from the network, process it and share it through a standard API. The client information gathered by ALE can be used for analyzing a client's internet behavior for business such as shopping preferences.

ALE is delivered as an open virtual appliance (OVA) file supported on VMware ESX/ESXi 5.x. This OVA file can be deployed using VMware vSphere. This procedure above automatically creates a virtual machine (VM) with the following specifications:

- Guest Operating System = Linux, Version = Centos 6 64 bit
- Number of Virtual Sockets = 1, Number of cores per virtual socket = 2
- Memory Configuration = 6 GB
- Number of NICs = 1

When deploying the OVA file, you can size the VM based on Aruba Networks, Inc. technical support recommendations. Additionally, after deployment, you can change the VM CPU, memory, and hard drive if you run into any performance issues.

Setting Up and Installing ALE

You can set up and install ALE 1.2 on a virtual machine or a standard server.

Only ALE 1.2 and later can be installed on a standard server.

Installing ALE on a Virtual Machine

You can set up the VM using the Ova file.

The following steps describe how to set up ALE using the OVA file.

1. Save the downloaded OVA file in a location accessible by the vSphere application.
2. Launch vSphere. In the vSphere application:
   a. Navigate to **File >Deploy OVF Template**.
   b. Select the downloaded ALE OVA file.
   c. Click **Next**.
   d. Accept the end-user license agreement to deploy the OVA file.
   e. Verify installation details and click **Next**.
   f. [Optional] This installation screen allows you to change the name of the installation file. Enter a new name, or skip this step to keep the default "ALE-1.1.0.0-xxx".

g. Click **Next** .

h. Select the datastore where this VM will be deployed, based on your ESX/ESXi setup. The disk space for the VM will be allocated in this datastore, and the format should be left to the default “Thick Provision Lazy Zeroed” setting.

i. Click **Next**.

j. Verify the final installation settings are accurate. f you need to make any changes, click the Back button and update the settings. Otherwise, click **Finish**.

3. If you want to change the default VM settings for CPU, memory or hard disk, do that before powering on.
4. On the login screen, log in as the admin user with the default admin user ID and password: User ID= root, password = admin

You must change the default password after logging in the first time.

5. Configure the network interface based on your local LAN settings, ALE needs a static IP address.

Installing ALE 1.2 on a Standard Server

The following steps describe how to set up and install ALE1.2 on a standard server.

1. Download the ISO image file and burn the image onto a DVD.
2. Install the DVD into the standard server. Continue with the installation process.
3. After the installation is complete, remove the DVD and restart the system.
4. On the login screen, log in as the admin user with the default admin user ID and password: User ID= root, password = admin

You must change the default password after logging in the first time.

5. Configure the network interface based on your local LAN settings, ALE needs a static IP address.

Setting Up ALE

To set up ALE, do the following:

- Configure the controller from which the ALE receives updates
- Configure the Airwave server from which ALE downloads floor maps
- Configure communication between the ALE and Meridian/Analytics Engines

Configuring the Controller and the Airwave Server with ALE

Follow the procedure below to set up the ALE with controller and AirWave settings. For controller setup, configure the controller from which the ALE receives updates. For the Airwave server setup,configure the Airwave server from which ALE is required to download the floor maps.

You can configure setup from the command line or the WebUI.

ALE requires a *read-only* controller user name and password. Do not use an administrator user name or password.

In the WebUI

NOTE

For configuration, the WebUI is not enabled by default.

1. Log onto ALE as root and run the following command:
   ```
   /opt/ale/bin/htpasswd.py -b /opt/ale/html/.htpasswd <userid> <password
   ```

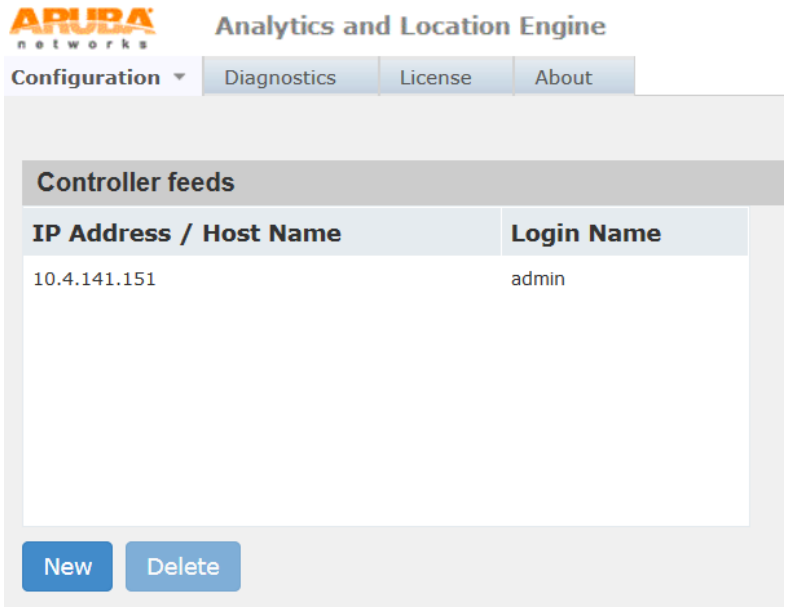
where `<userid>` and `<password>` are the new user ID and password that you want to create. You can pick any username.

2. Open your browser and type the following URL:
   ```
   http://<ale IP Address>
   ```
   The ALE UI page displays with the **Configuration** tab on top.
3. From the drop-down Configuration tab menu, select either **Controller feeds** or **Airwave feeds** to configure a controller or an Airwave server, respectively.

NOTE

Make sure that the correct floor plan with accurate AP locations is uploaded to the Airwave server before configuring Airwave on the ALE.

Figure 1 *ALE UI Configuration tab*

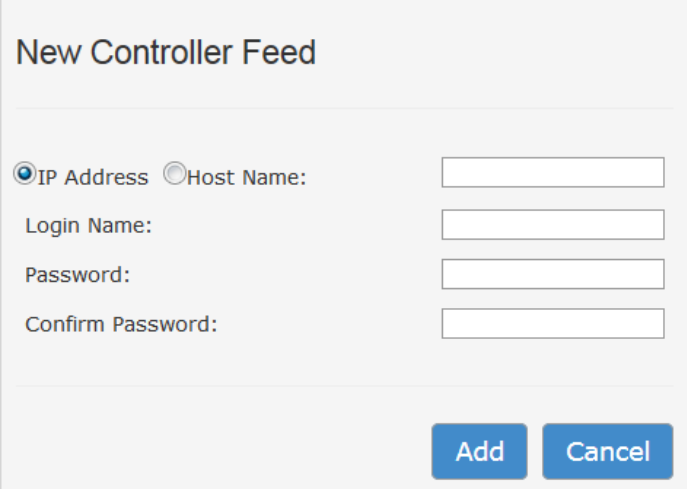


4. The **New Controller Feed** or **New Airwave Server Feed** dialog displays.
5. Enter the controller or Airwave server IP address or host name, login name, and password.

NOTE

Enter a read-only controller username and password. Do not use and administrator username or password.

Figure 2 *ALE UI Configuration tab - New Controller Feed dialog box*



6. Click **Add** and then **Apply**.

In the Command Line

1. Access the VM and navigate to the directory **/opt/ale/etc**.
2. Edit the locationserver.conf text file using any text editor.
   a. Add the following controller information under the **<controllers>** tag.
   - **controller hostname**="<controller-ip-addr>"
   - **port**="4343"

NOTE

The port number is fixed.

   - **username**="< controller-readonly-username>"
   - **password**="< controller-readonly-username>"
   - **cacert**="<cert>" (this parameter is optional, and can be left blank.)

   For example:

```
<controllers>
<controller hostname="10.1.50.3" port="4343" username="admin" password="admin" cace
rt=""/>
```

   b. Add the following AirWave information under the **<sitefetch>** tag:
   - airwave host="<AirWave-ip-addr>""
   - **username**="<username>"
   - **password**="<password>"

Passwords must be escaped properly for XML. For example if your password is **r&&t**, then it must be written as **r&&t** . In future, this file will contain the encrypted passwords

NOTE

The AirWave username/password must be a user which has access to the VisualRF in Airwave (either read-only or read-write) and the Access Points located on that floor. See your AMP admin guide for details on user roles and privileges.

3. After configuring the controller and ALE server from the CLI, restart the location server module of ALE using the following command to initiate the initial HTTP fetch from the controller and to download floor maps from Airwave:

```
[root@ale ~]# /etc/init.d/locationserver restart
```

Configuring the Meridian Analytics Engines with ALE

Each ALE instance has a unique ID that is carried with every message it sends. The unique ID allows Meridian to distinguish the traffic that arrives from each customer and is used when configuring communication with the Meridian Analytics Engines.

In the ALE UI, select the **About** tab in the ALE U to view the ALE ID.

Configuring the Controller

To configure your Arubacontrollers to send information from the controllers to ALE complete the below steps. ALE processes automatically start at system boot time.

After configuration changes are complete, you *must* restart the ALE.

1. Access the controller CLI in enable mode.
2. Run the CLI command:

```
(host) (config)#mgmt-server type ale primary-server <ALEIP>
profile default-ale
```

Starting the ALE Server

1. ALE processes are automatically started at system boot time. After the configuration changes are done, you must restart the processes.
2. Access the VM and navigate to the directory **/etc/init.d**.
3. Run these commands:

   ```
   ./iap-webapp restart
   ./ale-webapps restart
   ./locationserver restart
   ```

   The VM displays a message indicating that the server has started.

About the ALE Licenses

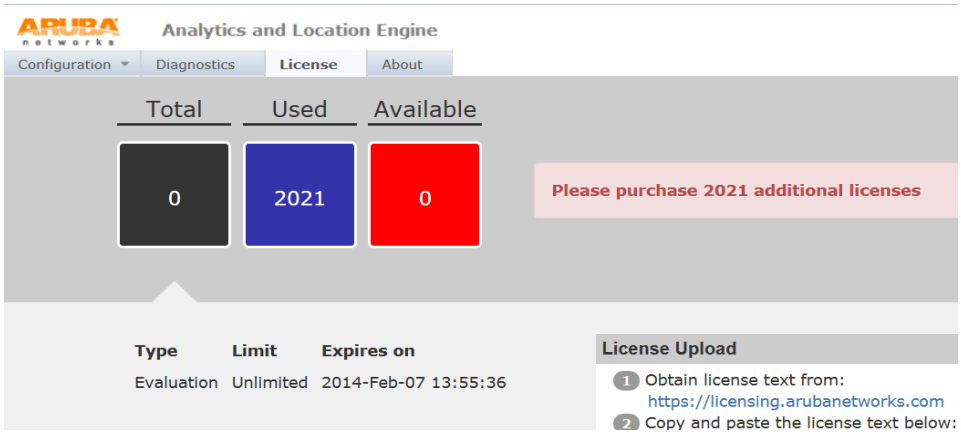
ALE ships with an evaluation license that expires after 90 days. You can use an unlimited number of APs during the evaluation license time period.

Viewing your Evaluation License

To view your evaluation license select the **License** tab in the ALE UI.

This tab indicates the license type (evaluation or permanent), total amount of licenses, number of licenses used and the remaining number available. Each AP requires its own license. You can use an unlimited number of APs with an evaluation license, but once the evaluation license expires you must buy a permanent license.

Figure 3 *ALE UI License tab - Evaluation license*

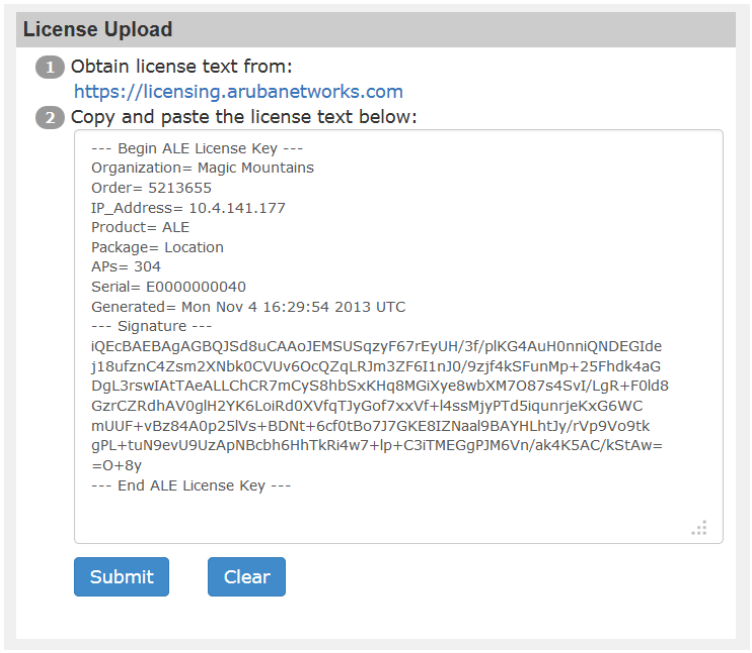


Obtaining a Permanent License

To obtain a permanent license:

1. On the ALE UI **License** tab, select the URL link https://licensing.arubanetworks.com to request a license.
2. Paste the ALE license key you receive by email into the license field on the **License** tab.

Figure 4 *ALE License tab - license key*



3. Click **Submit**. If the license is valid a confirmation message appears.

Integrating ALE with Instant AP

ALE supports integration with IAP. The ALE server acts as a primary interface to all third-party applications and the IAP sends client information and other status information to the ALE server. To integrate with ALE, the ALE server address must be configured on an IAP.

Enabling IAP for ALE Support

To enable an IAP for ALE support, run these commands:

```
(host) configure terminal
(host) (config)  ale-server <server:port>
(host) (config)  exit
(host) commit apply
```

Syntax

Command/Parameter	Description
`ale-server <server:port>`	Allows you to specify the Fully Qualified Domain Name (FQDN) or IP address of the ALE server.
`no…`	Removes the specified configuration parameter.

Example

The following example enables ALE on an IAP:

```
(host)(config)# ale-server AleServer1:8088
```

Interval Configuration

To configure the interval at which an IAP sends data to the ALE server, use this command:

```
ale-report-interval <seconds>
no…
```

Syntax

Command/Parameter	Description	Range	Default
`ale-report-interval <seconds>`	Configures an interval at which the Virtual Controller can report the IAP and client details to the ALE server.	6-60 seconds	30
`no…`	Removes the specified configuration parameter.	–	–

Example

The following example configures the ALE server details:

```
(host)(config)# ale-report-interval 60
```

About ALE and Firewalls

The following table displays firewall configuration information for ALE communication types.

Table 3: *Firewall Configuration for ALE*

Communication	Port	Protocol	Notes
Controller > ALE	8211	PAPI	For AMON feed from controller

Communication	Port	Protocol	Notes
Instant VC > ALE	8088	HTTPS	Port is configurable
ALE > Controller	4343	HTTPS	For fetching API data
ALE > Airwave	443	HTTPS	Fetch floorplan data from Airwave
ALE > Meridian	443	SSL	Location data to Meridian
ALE > Cloud	7779	SSL	ZeroMQ feed
HTTP Client > ALE	8080	HTTP	Restful APIs
User > ALE	80	HTTP	ALE GUI

About Anonymization

Direct use of a user's personal unique identity information, which exists within specific message data fields on the amon feed, raises privacy issues within ALE.

Serious privacy issues arise when personal network and mobile device identification information, which is saved by ALE, is shared with outside applications. Therefore, information such as the device's MAC address, acquired (associated) IP Address (or a history of it), a username, and other similar personal or sensitive and unique identity information must be anonymized and the anonymization cannot be reversed.

Anonymization Basics

- Anonymization is a configurable option in ALE and is activated by default. Once activated, all subscribers get anonymized data.
- The following fields are anonymized for both Publish/Subscribe and REST APIs: mac_addresses, ip_addresses and usernames.
- When anonymization is activated, only the getALL option for all REST queries is supported. Filtering by MAC, IP address or username will return an error.
- ALE uses a salted keyed SHA-1 hashing algorithm to generate the anonymized fields.
- Anonymization can be turned off by defining a new property in /etc/nbapi.properties. The property is named "ale.anonymization" and accepts Boolean values "true" (default) and "false".

Sample Anonymization On/Off Message Output

For example, a Station query to a North Bound API displays the following message output when ALE anonymization is turned *on*:


```
"Station_result":[
{
"msg":{
"role":"Aruba-Employee",
"bssid":{
"addr":"6CF37FEC1110"
},
"device_type":"iPad",
"hashed_sta_eth_mac":"041CB396A0844FE3BF3A6F22B7475ED037BD972B",
"hashed_sta_ip_address":"34A71F00D8A61467739009283665CE47CEC21E1A"
},
"ts":1393536217
}
```

```
]
}
```


When anonymization is turned *off*, the same Station query displays the following message output:

The red text appears in the message output file when anonymization is off.


```
"Station_result":[
{
"msg":{
"role":"Aruba-Employee",
"username":"jdoe",
"sta_eth_mac":{
"addr":"6482FFBB2A35"
},
"bssid":{
"addr":"6CF37FEC1110"
},
"sta_ip_address":{
"af": "ADDR_FAMILY_INET",
"addr": "10.100.239.186"
},
"device_type":"iPad",
"hashed_sta_eth_mac":"041CB396A0844FE3BF3A6F22B7475ED037BD972B",
"hashed_sta_ip_address":"34A71F00D8A61467739009283665CE47CEC21E1A"
},
"ts":1393536217
}
]
}
```


Changing the Salting so that Hash MAC Addresses Cannot be Traced

ALE 1.2 introduces a method (changing the salt used in the hashing algorithm) where MAC addresses are still collected, but you can change the salting so that the hash MAC addresses cannot be traced. You can do this by setting the salting schedule.

To change the salting schedule, follow these steps:

1. Open the `ale.hash.schedule` file, which is located in `/etc/nbapi.properties` file, to change the values to one of the following:

Table 4: *Salting Schedule*

Value	Meaning
Daily	Fire at midnight every day.
Weekly	Fire at midnight every Sunday
Monthly	Fire at midnight on the first day of every month.
Never	Never change the hash. *This is the default.*

2. Restart the ale-webapps and locationserver for the salting schedule to take effect.

The Analytics and Location Engine (ALE supports two types of APIs. One is a polling-based REST API, and the other is a publish/subscribe API based on Google Protobuf and ZeroMQ. This section describes the format of the information included in the API, the types of data each API can return, and the steps required to use these APIs to view ALE data.

- The REST based API supports HTTP GET operations by providing a specific URL for each query. Each query can be modified to include one or more parameters to refine the types of data returned by the query. For more information on ALE Polling APIs, see Polling APIs on page 19
- The publish/subscribe API is based on the ØMQ transport. A subscriber uses ØMQ client libraries to connect to the ALE and receive information from ALE asynchronously. For more information on this group of APIs, see Publish/Subscribe APIs on page 26

Polling APIs

The Representational State Transfer (REST) polling-based API supports HTTPS GET operations by providing a specific URL for each query. Refer to the following sections for details about each of the Polling APIs supported by the ALE. Output format is in Google Protobuf, or JSON.

Stations API

The Stations API displays information about clients associated to the controllers that send information to ALE. Every associated and authenticated user on the wireless network is represented by a station object. The XML response to this query type can display the following types of information about a station.

Station API queries use the URL syntax:

```
http://localhost:8080/api/v1/station
```

Table 5: *Station API Output*

Output Parameter	Definition
sta_mac_address	MAC address of the client station.

Output Parameter	Definition
username	Client's user name, If a user name is not defined, this parameter will display the client's IP address.
role	Name of the user role currently assigned to the client. This is applicable to only authenticated users..
bssid	BSSID that to which this station is associated.
device_type	Client's device type. The ALE can detect and return information for any of the following client device types. • Windows 7 • iOS devices
sta_ip_address	IP address of the client.
hashed_sta_eth_mac	This is the anonymized MAC address of the station, available when anonymization is enabled.
hashed_sta_ip_address	This is the anonymized IP address of the station, available when anonymization is enabled.

```
http://10.4.250.10:8080/api/v1/station
```

This query displays output similar to the example below.

```
{
"Station_result": [
{
"msg": {
"sta_eth_mac": {
"addr": "F4F15AA2B8E0"
},
"username": "Vjammula",
"role": "Aruba-Employee",
"bssid": {
"addr": "D8C7C888D0D0"
},
"device_type": "iPhone",
"sta_ip_address": {
"af": "ADDR_FAMILY_INET",
"addr": "10.11.9.248"
},
"hashed_sta_eth_mac": "09097EA8F4DACD5A55F3A9D2F456EFE557D35F09",
"hashed_sta_ip_address": "436738A08110E88906F8A14CCEF66949A3DBAE01"
},
"ts": 1381977108
},
}
```

Access Point API

The access points (AP) API displays information about APs that terminate on the controllers configured to send information to ALE.

AP API queries use the URL syntax:

`http://localhost:8080/api/v1/access_point`

The XML response to this query type can display the following types of information about an AP.

Table 6: *AP API Output*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP.
ap_name	Name of the AP. If the AP does not have a name, this API will return the IP address of the AP.
ap_group	Name of the AP group to which the AP is assigned.
ap_model	Model number of the AP.
depl_mode	AP's deployment mode strings, displayed as an integer that corresponds to a mode type.
ap_ip_address	IP address of the AP.

```
http://localhost:8080/api/v1/access_point
```

This query displays output similar to the example below.

```
{
"Access_point_result": [
{
"msg": {
"ap_eth_mac": {
"addr": "D8C7C8C0C7BE"
},
"ap_name": "1344-1-AL5",
"ap_group": "1344-hq",
"ap_model": "135",
"depl_mode": "DEPLOYMENT_MODE_CAMPUS",
"ap_ip_address": {
"af": "ADDR_FAMILY_INET",
"addr": "10.6.66.67"
}
},
"ts": 1382046667
}]}
```

Application API

The Application API is not available for Instant AP (IAP) deployment.

Queries using the Application API return a list of applications classified by the controller. The response to this query type can display the following types of information.

Client Application API queries use the URL syntax:

```
http://localhost:8080/api/v1/application
```

Table 7: *Application API Output*

Output Parameter	Definition
app_id	A unique number assigned to this application by the controller.
app_name	Name of the application as assigned by controller.

```
http://10.4.250.10:8080/api/v1/application
```

This query displays output similar to the example below.

```
{
"Application_result": [
{
"msg": {
"app_id": 50331801,
"app_name": "Smarter Balanced Testing"
}
}]}
```

Destination API

The Destination API is not available for IAP deployment.

Queries using the Destination API return a list of client destinations; IP addresses to which traffic is sent. The response to this query type can display the following types of information.

Destination API queries use the URL syntax:

```
http://localhost:8080/api/v1/destination
```

Table 8: *Destination API Output*

Output Parameter	Definition
dest_ip	IP address of a client receiving traffic through the network.
dest_name	Name of client destination.
dest_alias_name	An alias name assigned to this destination in controller.

```
http://localhost:8080/api/v1/destination
```

This query displays output similar to the example below.

```
{
"Destination_result": [
{
"msg": {
"dest_ip": {
"af": "ADDR_FAMILY_INET",
"addr": "98.175.77.106"
},
"dest_name": "adserving.autotrader.com",
"dest_alias_name": "autotrader"
}
}]}
```

Campus API

Campuses contain buildings which have individual floor maps. These maps must be defined using AirWave or Visual RF before they are imported into the ALEconfiguration during setup and initialization. If the map is changed, it must be reimported for the changes to take affect.

Campus API queries use the URL syntax:

```
http://localhost:8080/api/v1/campus
```

Table 9: *Campus API Output*

Output Parameter	Definition
campus_id	ID number identifying a specific campus location.
campus_name	Name given to a campus location

```
 http://localhost:8080/api/v1/campus
```

This query displays output similar to the example below.

```
{
"Campus_result": [
{
"msg": {
"campus_id": "6F9DEC79839D458B9F148D16A46A353E",
"campus_name": "GAP"
},
"ts": 1382046667
},]}
```

Building API

This API returns information about the buildings within each campus structure. These maps must be defined using AirWave or Visual RF before they are imported into the ALEconfiguration during setup and initialization. If the map is changed, it must be reimported for the changes to take affect. Each building name must be unique within a campus. However, other campuses can also have a building with the same building name.

Building API queries use the URL syntax:

```
http://localhost:8080/api/v1/building
```

Table 10: *Building API Output*

Output Parameter	Definition
building_id	ID number identifying a specific campus building.
building_name	Name given to a building campus building.
campus_id	ID number identifying a specific campus location.

```
http://localhost:8080/api/v1/building
```

This query displays output similar to the example below.

```
{
"Building_result": [
{
"msg": {
"building_id": "83393A922FB249C1929B95393A2AAFDA",
"building_name": "3600-RFBOX",
"campus_id": "ECDDE4535C8E4723B8AF849B3F86E7BF"
},
"ts": 1382046667
}]}
```

Floor API

This API retrieves floor definitions for each building in each campus. Campuses contain buildings which have individual floor maps. These maps must be defined using AirWave or Visual RF before they are imported into the ALEconfiguration during setup and initialization. If the map is changed, it must be reimported for the changes to take affect. Each floor name must be unique within a building. However, other buildings can also have a floor with the same floor name.

Floor API queries use the URL syntax:

```
http://localhost:8080/api/v1/floor
```

Table 11: *Floor API Output*

Output Parameter	Definition
floor_id	ID number identifying a specific building floor.
floor_name	Name given to a building floor.
floor_latitude	Latitude coordinate of top left corner of this floor, as defined in Visual RF.
floor_longitude	Longitude coordinate of top left corner of this floor, as defined in Visual RF.
floor_img_path	URL path to retrieve the background image for this floor.
floor_img_width	Floor width in feet.
floor_img_length	Floor length in feet.
building_id	ID number identifying the building that contains this floor.

```
http://localhost:8080/api/v1/floor
```

This query displays output similar to the example below.

```
{
"Floor_result": [
{
"msg": {
"floor_id": "1C48EF7D78DC4B948F1A15D7CD14FDED",
"floor_name": "Floor 1",
"floor_latitude": 0,
"floor_longitude": 0,
"floor_img_path": "/images/plan/img_1c48ef7d-78dc-4b94-8f1a-15d7cd14fded.jpg",
"floor_img_width": 246.33,
"floor_img_length": 249.92,
"building_id": "DAD3A7092AC04AA4B2F5DAF266EBD81B"
},
"ts": 1382046667
}]}
```

Presence API

This API retrieves presence objects.

Presence API queries use the URL syntax:

```
http://localhost:8080/api/v1/presence
```

The Presence API supports several different query parameters. Focus your query on one or more individual presence objects by including the following query parameters in your polling request. Queries that include multiple parameters must format the query with an ampersand (&) symbol between each query parameter.

Table 12: *Presence API Query Parameters*

Query Parameter	Definition
associated	Returns a boolean values true or false. True, if the MAC address is associated or False, if the MAC address is not associated anymore.

The output be filtered to include information only for the records that match your request.

Table 13: *Presence API Output*

Output Parameter	Definition
stat_eth_mac	MAC address of the client as seen by the Access Point.
bool_associated	Indicates whether this client is associated or not.
hash_stat_eth_mac	When anonymization is enabled, only this field is returned instead of the real MAC address.

```
http://localhost:8080/api/v1/presence
```

This query displays output similar to the example below.

```
{
"Presence_result": [
{
"msg": {
"sta_eth_mac": {
"addr": "00FF00043DFF"
},
"associated": true,
"hashed_sta_eth_mac": "B9081DE2290A1670307F127D07935F788C7614DE"
},
"ts": 1381871586
}]}
```

Location API

This API retrieves Retrieves historical location objects for a specific MAC client. The last 1000 historical locations are stored for each MAC address. This API also publishes a location event if ALE receives an RSSI reading from a single AP for a station. To publish a single AP location, enable recipe 22 as it is disabled by default.

Location API queries use the URL syntax:

```
http://localhost:8080/api/v1/location?sta_eth_mac=AA:BB:CC:DD:EE:FF
```

The Location API only supports querying by MAC address.

Table 14: *Location API Query Parameters*

Query Parameter	Definition
stat_eth_mac	Returns presence objects in context of a specific MAC address. For example, AA:BB:CC:DD:EE:FF.

The output will be filtered to include information only for the records that match your request.

Table 15: *Location API Output*

Output Parameter	Definition
sta_eth_mac	MAC address of the client.
sta_location_x	X coordinate for location (in feet).
sta_location_y	Y coordinate for location (in feet).
error_level	This is experimental, this value indicates the error level associated with this location calculation.
associated	Whether this client is associated.
campus_id	ID number identifying the campus.
building_id	ID number identifying the building.
floor_id	ID number identifying a campus building.
hashed_sta_eth_mac	Anonymized value of the MAC address.
loc_algorithm	Indicates how the (x,y) coordinates are populated for the message.

This query displays output similar to the example below.

```
{
"Location_result": [
{
"msg": {
"sta_eth_mac": {
"addr": "00FF00043DFD"
},
"sta_location_x": 35.035,
"sta_location_y": 251.61499,
"error_level": 214,
"campus_id": "A491E73EA7D34DEBA876AA667CB8353B",
"building_id": "A6648B8087954F16B5562C6088434CDE",
"floor_id": "B3D65D36E5D449CAA483000000000016",
"hashed_sta_eth_mac": "6C4157F77D50D0C7C052B5094B6621B47BADC969"
"loc_algorithm": "ALGORITHM_AP_PLACEMENT"
},
"ts": 1381954509
}]}
```

Publish/Subscribe APIs

ALEpublish/subscribe API uses ØMQ client libraries to allow network administrators to connect to the ALE and subscribe to selected topics. Once a user has subscribed to a topic, ALE starts publishing messages on these topics, and sends them to the subscribers.

ALE manages access rights to personally identifiable information (PII) such as MAC addresses, client user names, and IP addresses of network devices by removing this sensitive information from data feeds to subscribers who do not have the appropriate credentials. All messages are encoded using Google Protocol Buffer and specified using a .proto file. Network application developers, developing applications processing this data, use this .proto file and the

protocol buffer compiler,(protoc), to generate the message parsing code in the language of your choice (such as C++, Java or Python).

Events

The Northbound API (NBAPI) publishes messages as events. An event message is the only message type NBAPI will publish. The NBAPI embeds other message types depending on the event.

The station is removed from the ALE table if the controllers also remove the station from it's user table. The user idle-timeout for removal is 5 minutes by default and is configurable by using "aaa timers idle-timeout x" where x is the number of minutes for the user to be idled out of the system.

The event protobuf schema is as follows:

```
message nb_event {
enum event_operation {
OP_ADD = 0;
OP_UPDATE = 1;
OP_DELETE = 2;
}
enum license_info {
hb_Dhak                  = 10;
hb_ThresholdXNotice      = 20;
hb_ThresholdOkNotice     = 21;
hb_LimExceeded           = 31;
hb_EvalStarted           = 41;
hb_NewLimExceeded        = 51;
hb_EvalDone              = 61;
hb_ALSOnline             = 71;
hb_ALSDieing             = 81;
hb_CODE_RED_BLOCKIT      = 91;
}
optional uint64 seq = 1;
optional uint32 timestamp = 2;
optional event_operation op = 3;
optional uint64 topic_seq = 4;
optional bytes source_id = 5;
//For now required license_info lic_info = 6 [default=hb_Dhak];
optional license_info lic_info = 6 [default=hb_Dhak];
// One of the following is populated depending on the topic
optional location location = 500;
optional presence presence = 501;
optional rssi rssi = 502;
optional station station = 503;
optional radio radio = 505;
optional destination destination = 507;
optional application application = 509;
optional visibility_rec visibility_rec = 510;
optional campus campus = 511;
optional building building = 512;
optional floor floor = 513;
optional access_point access_point = 514;
optional virtual_access_point virtual_access_point = 515;
optional geofence geofence = 516;
}
```

Global field definitions are as follows:

- lic_info: Current information about licensing status
- seq : Uniquely assigned global sequence number

- timestamp : Time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds when this event occurred
- op : Event operation code. Reflects the new object state. Possible values are: OP_ADD, OP_UPDATE, OP_DELETE
- topic_seq : Per topic uniquely assigned sequence number
- source_id : Random number of bytes used as a unique ID for the source ALE. This number is unique per ALE instance. The unique source_ids are persisted across reboots.

Presence

When a user is subscribed to the presence topic, the API publishes event messages with the optional sub message "presence." This message is sent as soon as an AP radio hears a selected client MAC address. If the client associates to the network, then the associated field is set to **true**, otherwise **false**.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "presence"
Protobuf schema message presence {
optional mac_address sta_eth_mac = 1;
optional bool associated = 2;
optional bytes hashed_sta_eth_mac = 3;
}
```

RSSI

This topic sends the RSSI value for a selected station at a specific point in time, as heard by an AP radio identified by radio_mac field.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "rssi"
Protobuf schema message rssi {
optional mac_address sta_eth_mac = 1;
optional mac_address radio_mac = 2;
optional uint32 rssi_val = 3;
optional bool associated = 4;
optional bytes hashed_sta_eth_mac = 5;
}
```

Location

The location topic sends updates for a specific station. This message will be sent as soon as the ALE calculates the location of an associated or unassociated client with a specified MAC address. The X and Y station location values in this topic indicate the number of feet that the station is away from the top left corner of the floor map..

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "location"
Protobuf schema message location {
optional mac_address sta_eth_mac = 1;
optional float sta_location_x = 2;
optional float sta_location_y = 3;
optional uint32 error_level = 7;
optional bool associated = 8;
optional bytes campus_id = 9;
optional bytes building_id = 10;
optional bytes floor_id = 11;
optional bytes hashed_sta_eth_mac = 12;
repeated bytes geofence_ids = 13;
optional algorithm loc_algorithm = 14;
}
```

Station

This station topic returns information about a user associated to a particular station MAC address. This message is only sent when a user authenticates to the network with a user name.

The station is removed from the ALE table if the controllers also remove the station from it's user table. The user idle-timeout for removal is 5 minutes by default and is configurable by using "aaa timers idle-timeout x" where x is the number of minutes for the user to be idled out of the system.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "station"
Protobuf schema message station {
optional mac_address sta_eth_mac = 1;
optional string username = 2;
optional string role = 3;
optional mac_address bssid = 4;
optional string device_type = 5;
optional ip_address sta_ip_address = 6;
optional bytes hashed_sta_eth_mac = 7;
optional bytes hashed_sta_ip_address = 8;
}
```

Access Point

This access point message is sent when a new access point (AP) is deployed in the network. The message will be sent as soon as the AP joins the network, regardless of whether the AP has been logically placed on a floor map. It will also be sent when an AP goes down and then comes back up and joins a controller.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "access_point"
Protobuf schema message access_point {
enum deployment_mode {
DEPLOYMENT_MODE_CAMPUS = 0;
DEPLOYMENT_MODE_REMOTE = 1;
}
optional mac_address ap_eth_mac = 1;
optional string ap_name = 2;
optional string ap_group = 3;
optional string ap_model = 4;
optional deployment_mode depl_mode = 5;
optional ip_address ap_ip_address = 6;
}
```

Radio

The radio topic sends information about each radio on a newly added AP.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "radio"
Protobuf schema message radio {
enum radio_mode {
RADIO_MODE_AP = 0;
RADIO_MODE_MESH_PORTAL = 1;
RADIO_MODE_MESH_POINT = 2;
RADIO_MODE_AIR_MONITOR = 3;
RADIO_MODE_SPECTRUM_SENSOR = 4;
RADIO_MODE_UNKNOWN = 5;
}
enum phy_type {
PHY_TYPE_80211B = 0;
PHY_TYPE_80211A = 1;
PHY_TYPE_80211G = 2;
```

```
}
enum ht_type {
HTT_NONE = 0;
HTT_20MZ = 1;
HTT_40MZ = 2;
HTT_VHT_20MZ = 3;
HTT_VHT_40MZ = 4;
HTT_VHT_80MZ = 5;
}
optional mac_address ap_eth_mac = 1;
optional mac_address radio_bssid = 2;
optional radio_mode mode = 4;
optional phy_type phy = 5;
optional ht_type ht = 6;
}
```

Virtual Access Point

The virtual access point topic messages are sent for each virtual AP (VAP) associated with a newly added AP.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "virtual_access_point"
Protobuf schema message virtual_access_point {
optional mac_address bssid = 1;
optional string ssid = 2;
optional mac_address radio_bssid = 3;
}
```

Destination

The Destination topic message is not available for IAP deployment.

The destination topic messages are sent when a new destination is classified by the controller firewall.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "destination"
Protobuf schema message destination {
optional ip_address dest_ip = 1;
optional string dest_name = 2;
optional string dest_alias_name = 3;
}
```

Application

The Application message is not available for IAP deployment.

This application message is sent when a new application is classified or otherwise recognized by the controller firewall.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "application"
Protobuf schema message application {
optional uint32 app_id = 1;
optional string app_name = 2;
}
```

Visibility Records

The Visibility Records message is not available for IAP deployment.

This visibility records message represents a unique associated station session for each client MAC for a given application or destination.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "visibility_rec"
Protobuf schema message visibility_rec {
enum ip_protocol {
IP_PROTOCOL_VAL_6 = 6;
IP_PROTOCOL_VAL_17 = 17;
}
optional ip_address client_ip = 1;
optional ip_address dest_ip = 2;
optional ip_protocol ip_proto = 3;
optional uint32 app_id = 4;
optional uint64 tx_pkts = 5;
optional uint64 tx_bytes = 6;
optional uint64 rx_pkts = 7;
optional uint64 rx_bytes = 8;
optional bytes hashed_client_ip = 9;
optional mac_address device_mac = 10;
optional bytes hashed_device_mac = 11;
optional string app_name = 12;
}
```

Campus

This campus message is sent when synchronization between ALE and updated VisualRF maps creates when a new campus.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "campus"
Protobuf schema message campus {
optional bytes campus_id = 1; // 16 bytes id
optional string campus_name = 2;
}
```

Building

This building message is sent when synchronization between ALE and updated VisualRF maps creates when a new building. Each campus can have multiple buildings, but a building can be located on only one named campus.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "building"
Protobuf schema message building {
optional bytes building_id = 1; // 16 bytes id
optional string building_name = 2;
optional bytes campus_id = 3;  // 16 bytes id;
}
```

Floor

This floor message is sent when synchronization between ALE and updated VisualRF maps creates when a new floor. Each building can have multiple floors, but a floor can be located on only one named building.

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "floor"
Protobuf schemamessage floor {
```

```
optional bytes floor_id = 1; // 16 bytes id
optional string floor_name = 2;
optional float floor_latitude = 3;
optional float floor_longitude = 4;
optional string floor_img_path = 5;
optional float floor_img_width = 6;
optional float floor_img_length = 7;
optional bytes building_id = 8; // 16 bytes id
optional float floor_level = 9;
optional string units = 10;
}
```

This topic discusses how to use the SSL bridge and provides troubleshooting tips.

About SSL Bridge

Integration with Meridian

The ALE offers a tight integration with the Meridian way-finding application. In order to couple this integration, the ALE provides location and user data to the Meridian cloud service.

To start the Meridian SSL tunnel, issue this command:

```
/etc/init.d/meridian-tunnel start
```

If you want to automatically start the tunnel on eventual reboot, issue this command:

```
/etc/init.d/meridian-tunnel install
```

To stop the tunnel:

```
/etc/init.d/meridian-tunnel stop
```

To uninstall the script:

```
/etc/init.d/meridian-tunnel uninstall
```

SSL bridge is used to encrypt and pass ALE data traffic to a ØMQ subscriber application hosted in the cloud.

The “nbapisslbridge” component is provided to simplify ALE deployments in data centers behind firewalls and NATs in which the Northbound consumer of information is an application hosted in a public cloud. In these cases, a connection back to the ALE is impossible from the cloud application, for example, an application running in Amazon EC2.

Using SSL Bridge

```
 Usage: ./nbapisslbridge [options]

Options:
-w Enable watchdog. Process will re-spawn when unexpectedly terminated
-f <endpoint>  Frontend endpoint to connect to , this is the IP address:port of ALE
-b <endpoint>  Backend endpoint to SSL connect to , this is the name/IP address:port of the cl
oud application
-c <ca path>   ALE file path or directory, this is the certificate of Cloud Application SSL se
rver
```

Example

To forward traffic from the local ALE server with the IP address 192.100.1.10 and port 7779 to an AWS EC2 instance at ec2.company.com port 4433 using a specific certificate.

```
nbapisslbridge -w -f 192.168.1.10:7779 -b ec2.company.com:4433 -c ./my-x509cert.pem
```

Note that the SSL bridge establishes the connection (socket connect) to the backend endpoint. In this configuration the ØMQ subscriber has to listen (socket bind) for the incoming connection. SSL bridge is designed to handle socket disconnections and retries to connect in a timely manner, thus enabling the subscriber to go up and down without being concerned about reconnection.

Troubleshooting

This section discusses common ALEdebugging, startup, and runtime issues that you can troubleshoot.

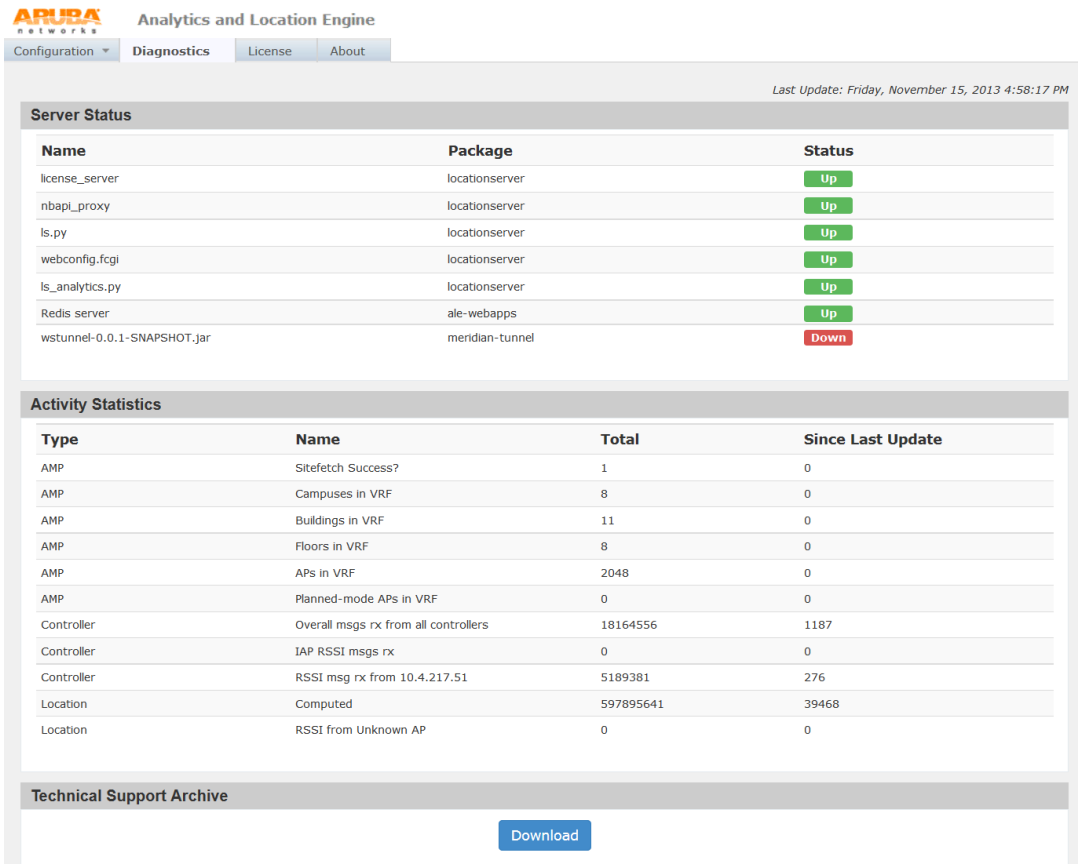
Viewing and Downloading the ALE Diagnostics

You can view ALE diagnostic information, such as server status and activity statistics, and download it as a file and send it to Aruba Tech Support.

To do this:

1. From the ALE UI, select the **Diagnostics** tab to view diagnostic data.
2. Click **Download** and save the file to your desktop.

Figure 5 *ALE UI Diagnostics tab*



This section includes sample code for the ZeroMQ application.

```
#include <stdio.h>
#include <string>
#include <iostream>
#include <iomanip>
#include <typeinfo>
#include <zmq.h>
#include "objects/schema.pb.h"
namespace gpb = google::protobuf;
namespace
{
const char* g_appName = NULL;
const char* const DEFAULT_ZMQ_ENDPOINT = "tcp://localhost:7779";
const char* const DEFAULT_ZMQ_SUB_FILTER = "";

void usage()
{
printf("\n");
printf("Usage: %s [options]\n", g_appName);
printf("Options:\n");
printf("  -e <endpoint>  ZMQ endpoint to connect/bind to. Default: %s\n", DEFAULT_ZMQ_ENDPOIN
T);
printf("  -f <filter>    Message filter to apply on a ZMQ_SUB socket. Default: %s\n",
strlen(DEFAULT_ZMQ_SUB_FILTER) ? DEFAULT_ZMQ_SUB_FILTER : "<empty>");
printf("  -b             Listen for ZMQ endpoint on port 7779. Default: connect\n");
printf("\n");
exit(1);
}
bool readMsgs(void* skt, zmq_msg_t& topic, zmq_msg_t& zmsg)
{
bool success = false;
int more = 0;
size_t more_size = sizeof(int);

if (zmq_msg_recv(&topic, skt, 0) >= 0)
{
/* Determine if more message parts are to follow */
if (zmq_getsockopt(skt, ZMQ_RCVMORE, &more, &more_size) == 0)
{
if (more)
{
if (zmq_msg_recv(&zmsg, skt, 0) >= 0)
success = true;
else
perror("zmq_msg_recv zmsg");
}
else
std::cerr << "No more Zmq message to read" << std::endl;
}
else
perror("zmq_getsockopt ZMQ_RCVMORE");
}
else
perror("zmq_msg_recv topic");
```

```
return success;
}

std::ostream& operator<<(std::ostream& os, const ce::nbapi::event& ev)
{
return os << ev.DebugString();
}
}
int main(int argc, char* argv[])
{
g_appName = argv[0];
std::string endpoint(DEFAULT_ZMQ_ENDPOINT);
std::string filter(DEFAULT_ZMQ_SUB_FILTER);
int c;
bool doBind = false;

while((c = getopt(argc, argv, "hf:e:b")) != -1)
{
switch (c)
{
case 'f':
if (optarg && optarg[0])
filter.assign(optarg);
break;
case 'e':
if (optarg && optarg[0])
endpoint.assign(optarg);
break;
case 'b':
doBind = true;
break;
default:
usage();
break;
};
}
void* ctx = zmq_ctx_new();
if (!ctx)
perror("zmq_ctx_new");
assert(ctx);

void* sub = zmq_socket(ctx, ZMQ_SUB);
if (!sub)
perror("zmq_socket");
assert(sub);
if (doBind)
{
endpoint.assign("tcp://*:7779");
printf("Attempting to 'bind' to endpoint: %s\n", endpoint.c_str());
if (zmq_bind(sub, endpoint.c_str()) != 0)
{
perror("zmq_bind");
assert(0);
}
}
else
{
printf("Attempting to 'connect' to endpoint: %s\n", endpoint.c_str());
if (zmq_connect(sub, endpoint.c_str()) != 0)
{
perror("zmq_connect");
assert(0);
```

```
}
}
printf("Connected to endpoint: %s\n", endpoint.c_str());
if (zmq_setsockopt(sub, ZMQ_SUBSCRIBE, filter.c_str(), filter.size()) != 0)
{
perror("zmq_setsockopt");
assert(0);
}
printf("Subscribed to topic: \"%s\"\n", filter.c_str());
zmq_msg_t topic;
zmq_msg_t zmsg;
if (zmq_msg_init(&topic) != 0)
{
perror("zmq_msg_init topic");
assert(0);
}
if (zmq_msg_init(&zmsg) != 0)
{
perror("zmq_msg_init zmsg");
assert(0);
}
std::string strTopic;
size_t cnt = 1;
ce::nbapi::event ev;

while (readMsgs(sub, topic, zmsg))
{
strTopic.assign(static_cast<const char*>(zmq_msg_data(&topic)), zmq_msg_size(&topic));
printf("[%zu] Recv event with topic \"%s\"\n", cnt, strTopic.c_str());
if (ev.ParseFromArray(zmq_msg_data(&zmsg), zmq_msg_size(&zmsg)))
{
std::cout << ev << std::endl;
}
else
{
printf("[%zu] Protobuf failed to parse event\n", cnt);
}
cnt++;
}
std::cout << "Cleaning..." << std::endl;

if (zmq_msg_close(&topic) != 0)
perror("zmq_msg_close topic");
if (zmq_msg_close(&zmsg) != 0)
perror("zmq_msg_close zmsg");

if (zmq_close(sub) != 0)
perror("zmq_close");

if (zmq_ctx_destroy(ctx) != 0)
perror("zmq_close");

return 0;
}
```