Development Kit Guideline

Index

Event Server - An IoT Platform

The Event Server is the core component of the AIGSG's IoT Platform. This database software is the platform's key element that takes care of the big amount of data (events) generated in the real world by the *things*, that is, doors, access devices, RFID tags and readers, any type of sensors and anything than can be digitally connected to a private network, VPN or the so called *internet cloud*.

The Platform is comprised of three central elements: an group of peripheral devices, gateways and the Event Server. As mentioned, the peripheral devices and sensors generate (receive) data from (to) the real world or process; this data flow is taken by the gateways, which are the interface to the network, VPN or Internet cloud that are just the means to reach out our platform final component: the Event Server. In any house, building, place, city and corporation generates millions of small pieces of information with a low value itself, but operators using regular software cannot handle this amount of events.

The Event Server organizes and processes automatically all events coming from the physical world, translated into the virtual world, following a specific and previously defined logic. In order to implement this *logic*, the Event Server is provided with a *script engine* module that allows the Administrator or programmer to establish cause-effect relations between incoming events (inputs) and outgoing events (outputs) from/ to the physical world. For instance, if a biometric reader registers a valid data (input), the access is granted (logic) for that person by opening the associated door (output).

Since the physical world is organized by itself in a hierarchical structure, that is, ***things*** (objects, assets) belong to specific ***places*** (facilities, buildings, corporations, etc), and also ***people*** are related to specific things and places, in one way or another. This situation is reflected in the Event Server through LDAP authentication. The ES works along with LDAP technology and software, such as, Active Directory, Open LDAP and others. Sensors, devices, gateways, locations, users, even scripts are *objects* that must be created and assigned to a hierarchical LDAP tree to exist in the ES.

In order to get benefit of the registered events allocated in the database, the Event Server is provided with an API interface, which enables data connection to Third Parties (Distribution Channels, Partners, re-sellers, etc) application software. These applications are the interface to the final users and operators (smartphones, tablets, operator stations, client PCs, and others).

Event Server Environment

The Event Server Environment is comprised by 2 core elements:

- AD/LDAP LDS Instance
- Event Server

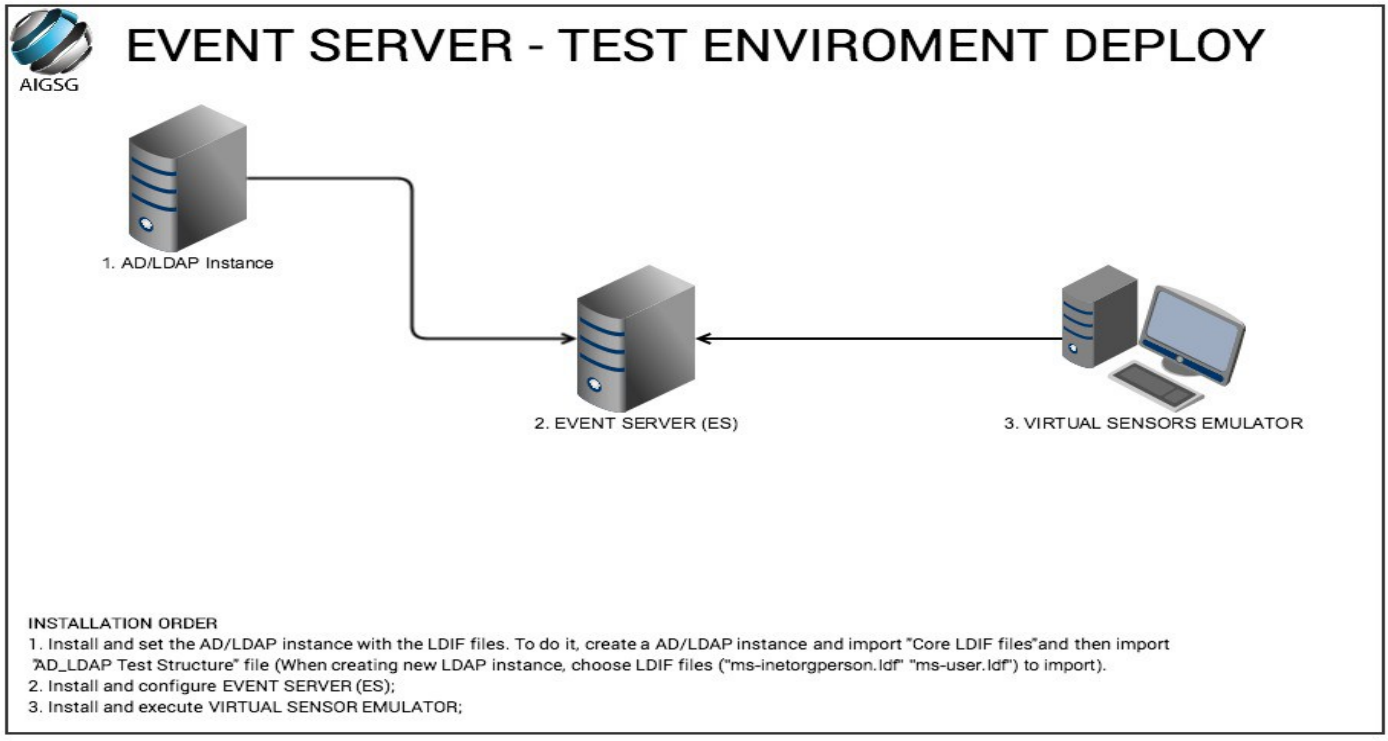
The AD/LDAP LDS Instance is responsible for all objects inside the IoT environment, like:

- Gateways
- Sensors
- Actuators
- People (Users)
- Locations (Places)
- Policies
- Scripts

The Event Server retrieves all the objects information from AD/LDAP LDS Instance and distribute them among the ES Nodes available in your IoT System.

Event Server – Test Environment Deploy

Let`s start the deployment of Event Server – Test Environment.
The image below show the installation order of this structure.



AD/LDAP LDS Instance

In order to install the LDS instance, execute the following steps:

1. Install Server roles AD and ADLDS in Windows Server.

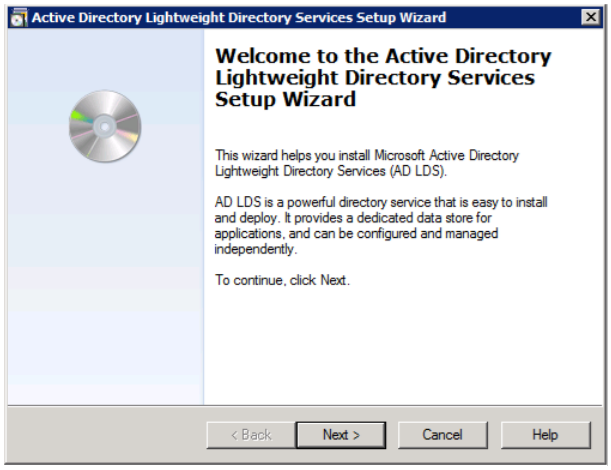
2. Disable Password Complexity for new users

Open Group Policy Management Console (Start / Run / GPMC.MSC), open the Domain, and right-click and Edit the "Default Domain Policy". Then dig into the "Computer Configuration", "Windows Settings", "Security Settings", "Account Policies", and modify the password complexity requirements setting.

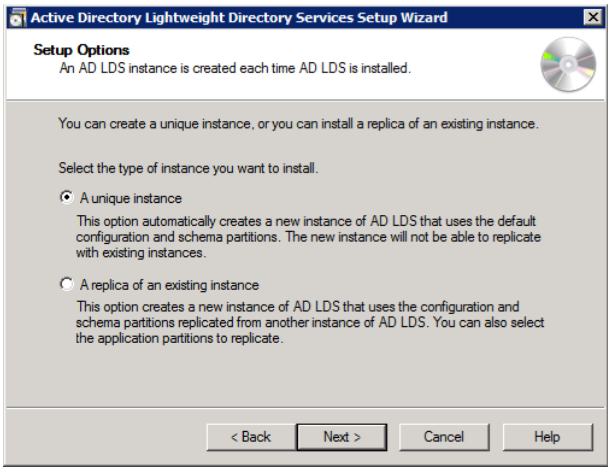
3. Installation of a LDS Instance

3.1. Execute “Active Directory Lightweight Directory Services Setup Wizard"

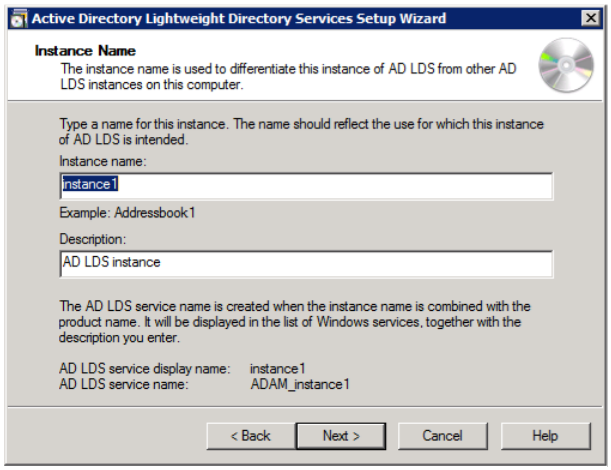
3.2. Follow the instructions below to create a new LDS instance.



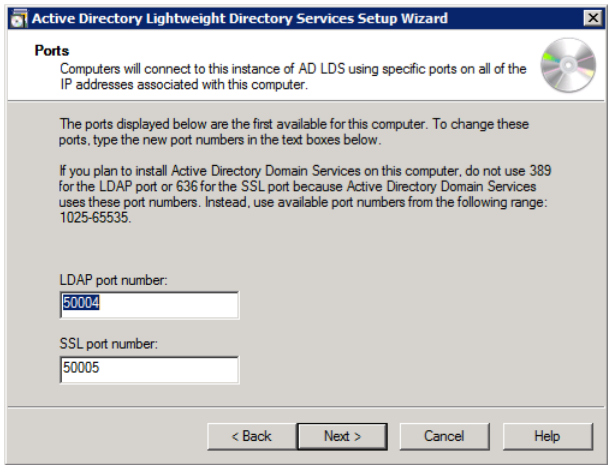
3.3. Select “A unique instance".



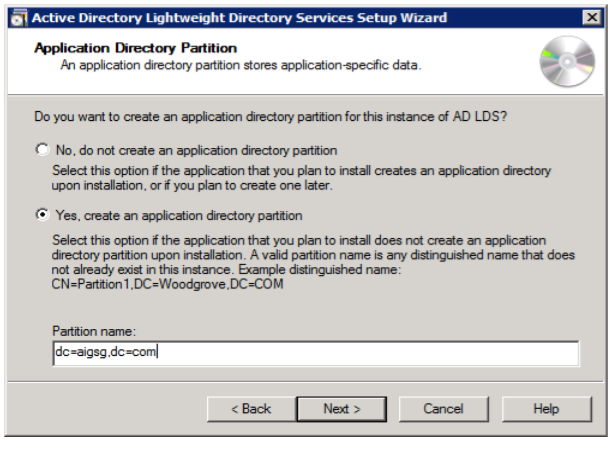
3.4. Create a name for the new LDS instance (This name is arbitrary).



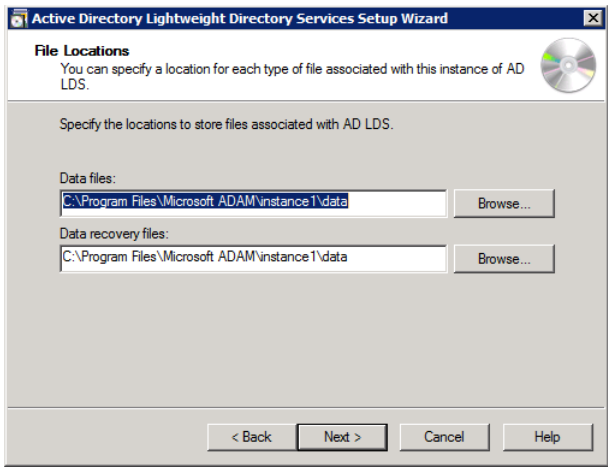
3.5. Define the ports of the LDS instance.



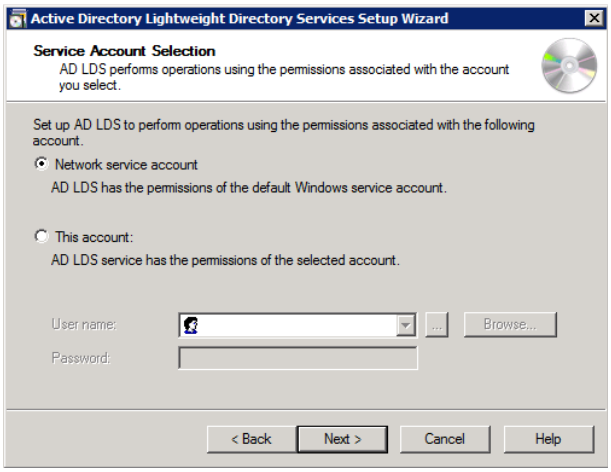
3.6. Select “Yes, create and application directory partition" and create a partition using “dc=aigsg,dc=com".



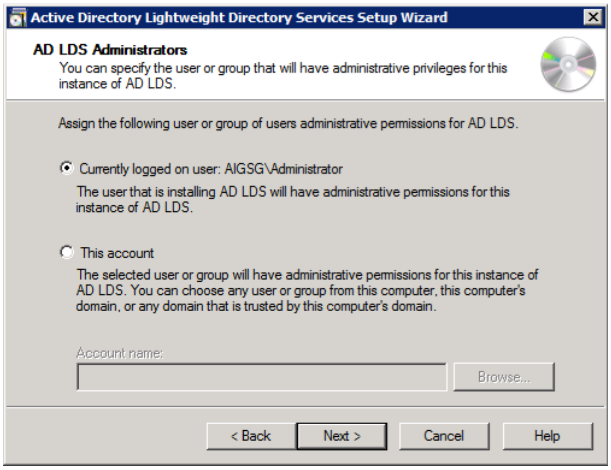
3.7. Select the path to store files associated with AD LDS.



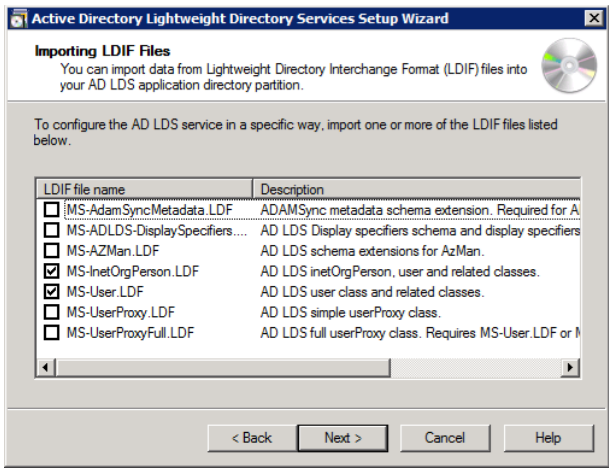
3.8. Select the Service account to perform operations in LDS instance.



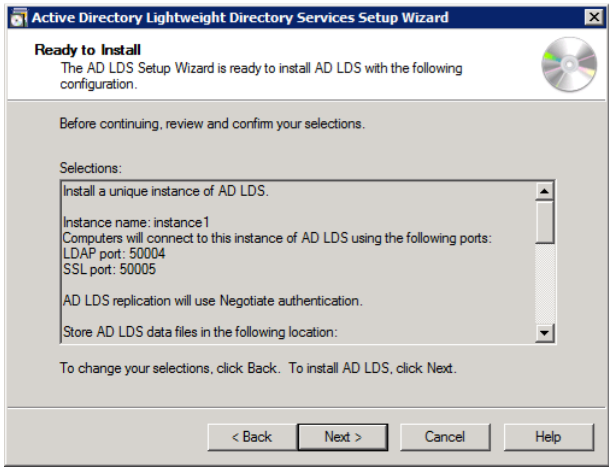
3.9. Select the AD LDS Administrator user for the LDS Instance.



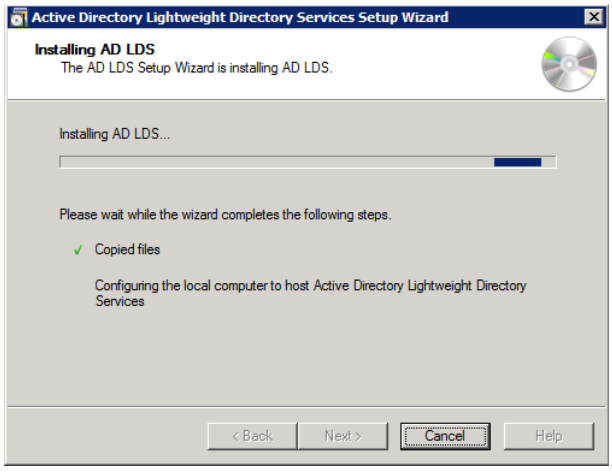
3.10. In “Importing LDIF Files”, select “MS-InetOrgPerson.LDF” and “MS-User.LDF”.



3.11. Confrm the information to create a new LDS Instance.



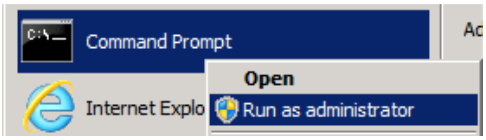
3.12. Wait until the end of creation of the new LDS instance.



4. LDS Instance deploy Scripts

4.1. Copy files from “ldap” folder to the AD/LDAP Server

4.2. Execute the “Command Prompt” as “Administrator”



4.3. In the CMD, access the folder “ldap” inside the computer

```
@echo off
echo Setting up schema...
set CONN=-s localhost:<port> -b <user> <domain> <password>
```

4.4. Change the “schema.bat” file and put the correct server information

<port> put LDAP Port Number.

<user> put a Username authorized to perform operations in LDS instance

<domain> put the domain of the username to perform operations in LDS instance

<password> put the password of the username to perform operations in LDS instance

E.g: set CONN=-s localhost:50004 -b Administrator AIGSG password

4.5. Execute the “schema.bat” file

4.6. Change the “data.bat” file and put the correct server information

```
@echo off
echo Initializing directory structure...
set CONN=-s localhost:<port> -h -b <user> <domain> <password>
```

<port> put LDAP Port Number.

<user> put a Username authorized to perform operations in LDS instance

<domain> put the domain of the username to perform operations in LDS instance

<password> put the password of the username to perform operations in LDS instance

E.g: set CONN=-s localhost:50004 -b Administrator AIGSG password

4.7. Execute the “data.bat” file

5. AD/LDAP Credentials

After the execution of “schema.bat” and “data.bat”, a new user will be available to change settings in LDS Instance and to login in Monitoring Center. Here are the credentials of this new user:

Username: administratoruser **Password:** passw0rd

Event Server

1. ES and Dependencies installation:

1.1. Install clean Ubuntu (15.04 “Vivid Vervet" 64 bits) with no additional packages chosen during installation (no Java, no database server, etc). Locale and language are all USA/English. Do not setup initial user as “eventsrv" – this one will be created later by .deb installer.

1.2. Install ES:

sudo dpkg -i eventsrv-1.0.1-0.deb

(at this point dpkg complaints about missing dependencies – ignore this message)

1.3. Run following command to force ES installation with its dependencies:

sudo apt-get install -f

1.4. Configure 'ES' database:

First, connect under an administrator user (which has CREATEDB and CREATEUSER privileges) and create new user and database which will be used by the Event Server. For that, run:

sudo -u postgres psql -f init.sql

Next, connect under "eventsrv" user and initialize database schema, for that run:

psql -h localhost -U eventsrv -f schema.sql

When you`re asked by a password in this step, use **eventsrv**

1.5. Install rabbitmq-server package:

sudo apt-get install rabbitmq-server

1.6. Configure RabbitMQ. Rabbitmqctl commands should be run under “rabbitmq" user:

sudo -u rabbitmq rabbitmqctl add_user eventsrv eventsrv

This command creates AMQP user for the Event Server;

sudo -u rabbitmq rabbitmqctl add_vhost eventsrv

This command creates AMQP virtual host;

sudo -u rabbitmq rabbitmqctl set_permissions -p eventsrv eventsrv ".*" ".*" ".*"

This command grants full access messaging permissions to previously created user within the virtual host.

2. ES initial Setup and service start

2.1. Rename eventsrv-sample.conf to eventsrv.conf (and update with correct settings).

Here is the content of the eventsrv.conf:

```
# Database settings
Database:
# Host name or address of the database server
Host: localhost
# Database name
Name: eventsrv
# Database user credentials
User: eventsrv
Password: eventsrv

# LDAP settings
Ldap:
# Host name or address of the LDAP server
Host: <IP Address of AD/LDAP Server>
# Port number of the LDAP server
Port: 389
# Root node of the organization structure
Root: ou=Organization,dc=aigsg,dc=com
# LDAP user credentials
User: admin@aigsg.com
Password: passw0rd

# AMQP settings
Amqp:
# Host name or address of the AMQP server Server
Host: localhost
# Virtual host name
VirtualHost: eventsrv
# AMQP user credentials
User: eventsrv
Password: eventsrv

# Primary settings
Primary:
# Path to installation directory
BaseDirectory: /var/lib/eventsrv
# Network address of this host
HostAddress: <IP Address of ES>
```

2.2 Run the server

```
# sudo service eventsrv start
```

ES license (250 objects)

To activate a license for 250 objects, follow the procedure below:

Copy the ES license files (250sensors.license) to:
/var/lib/eventsrv

then, go to folder /var/lib/eventsrv and run:
sudo chmod 777 250sensors.license

ES Virtual Sensors Emulator

1. Objective:

Virtual Sensor Emulator is a tool developed for test ES.

2. Pre requisites:

Computer with Ubuntu 15.04 64 bits installed.

3. Installation

3.1. Decompress the file “es-virtual-sensors-1.0.4.tar.gz"

3.2. Execute the following commands in order to update your Ubuntu installation:

sudo apt-get update

sudo apt-get upgrade

3.3. Install the dependencies:

sudo apt-get install libprotobuf-dev libboost-all-dev

3.4. Install QT opensource version 5.5

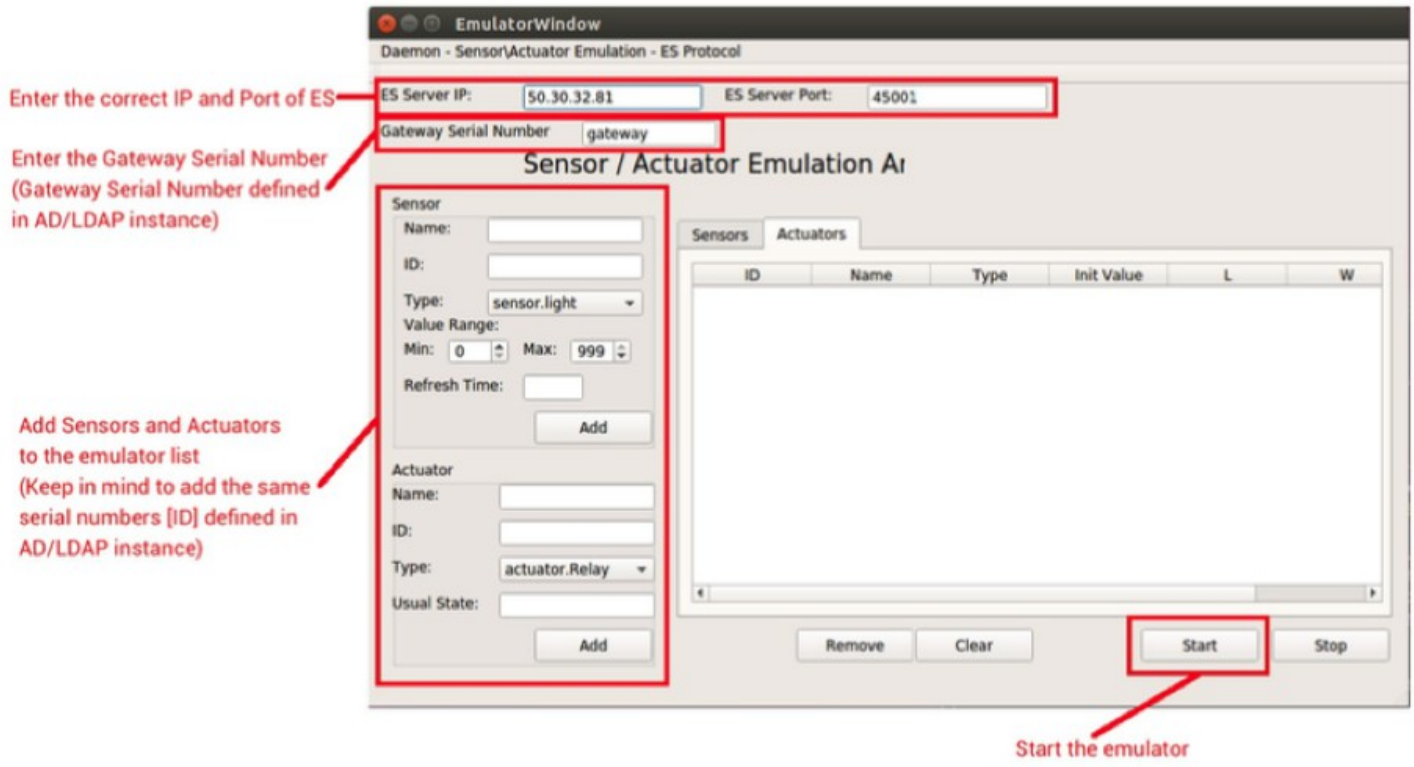
http://download.qt.io/archive/qt/5.5/5.5.0/qt-opensource-linux-x86-5.5.0.run

3.5. After install the updates and dependencies, run the script inside the decompressed folder “es- virtual-sensors":

./run.sh

4. How to use Virtual Sensors Emulator

Check the image below to verify a basic usage of Virtual Sensors Emulator:



1. Enter the correct IP and Port of the ES;
2. Enter the Gateway Serial Number (Same defned in AD/LDAP instance);
3. Add Sensors and Actuators (Keep in mind to use the same Serial numbers [ID] defned in AD/LDAP instance;
4. Start the emulator;

Scripts and Policies

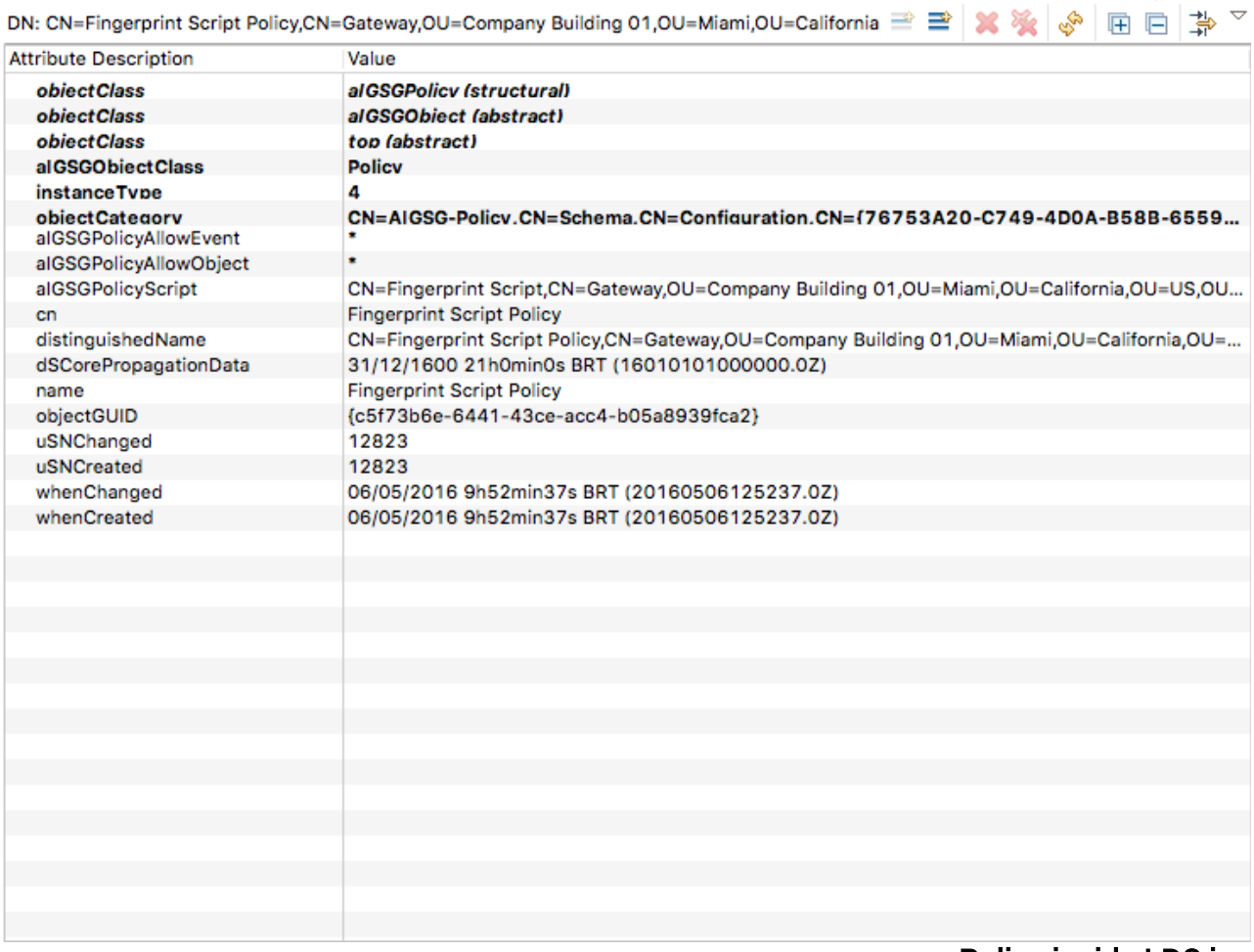
Event Server use Scripts and Policies inside LDS Instance in order to “automatize" and give permissions in the IoT environment.

Policies are in charge of the permissions inside LDS instance.

Scripts are in charge to give “Inteligence" and automation skills to the IoT environment.

1. Policies inside LDS instance

The policies inside the LDS instance allow the interaction between Scripts and Sensors.


DN: CN=Fingerprint Script Policy,CN=Gateway,OU=Company Building 01,OU=Miami,OU=California

Attribute Description	Value
objectClass	***aIGSGPolicy (structural)***
objectClass	***aIGSGObject (abstract)***
objectClass	***top (abstract)***
aIGSGObjectClass	**Policy**
instanceType	**4**
objectCategory	**CN=AIGSG-Policy.CN=Schema.CN=Configuration.CN={76753A20-C749-4D0A-B58B-6559...**
aIGSGPolicyAllowEvent	*
aIGSGPolicyAllowObject	*
aIGSGPolicyScript	CN=Fingerprint Script,CN=Gateway,OU=Company Building 01,OU=Miami,OU=California,OU=US,OU...
cn	Fingerprint Script Policy
distinguishedName	CN=Fingerprint Script Policy,CN=Gateway,OU=Company Building 01,OU=Miami,OU=California,OU=...
dSCorePropagationData	31/12/1600 21h0min0s BRT (16010101000000.0Z)
name	Fingerprint Script Policy
objectGUID	{c5f73b6e-6441-43ce-acc4-b05a8939fca2}
uSNChanged	12823
uSNCreated	12823
whenChanged	06/05/2016 9h52min37s BRT (20160506125237.0Z)
whenCreated	06/05/2016 9h52min37s BRT (20160506125237.0Z)

Policy inside LDS instance

The main attributes of a policy object are:

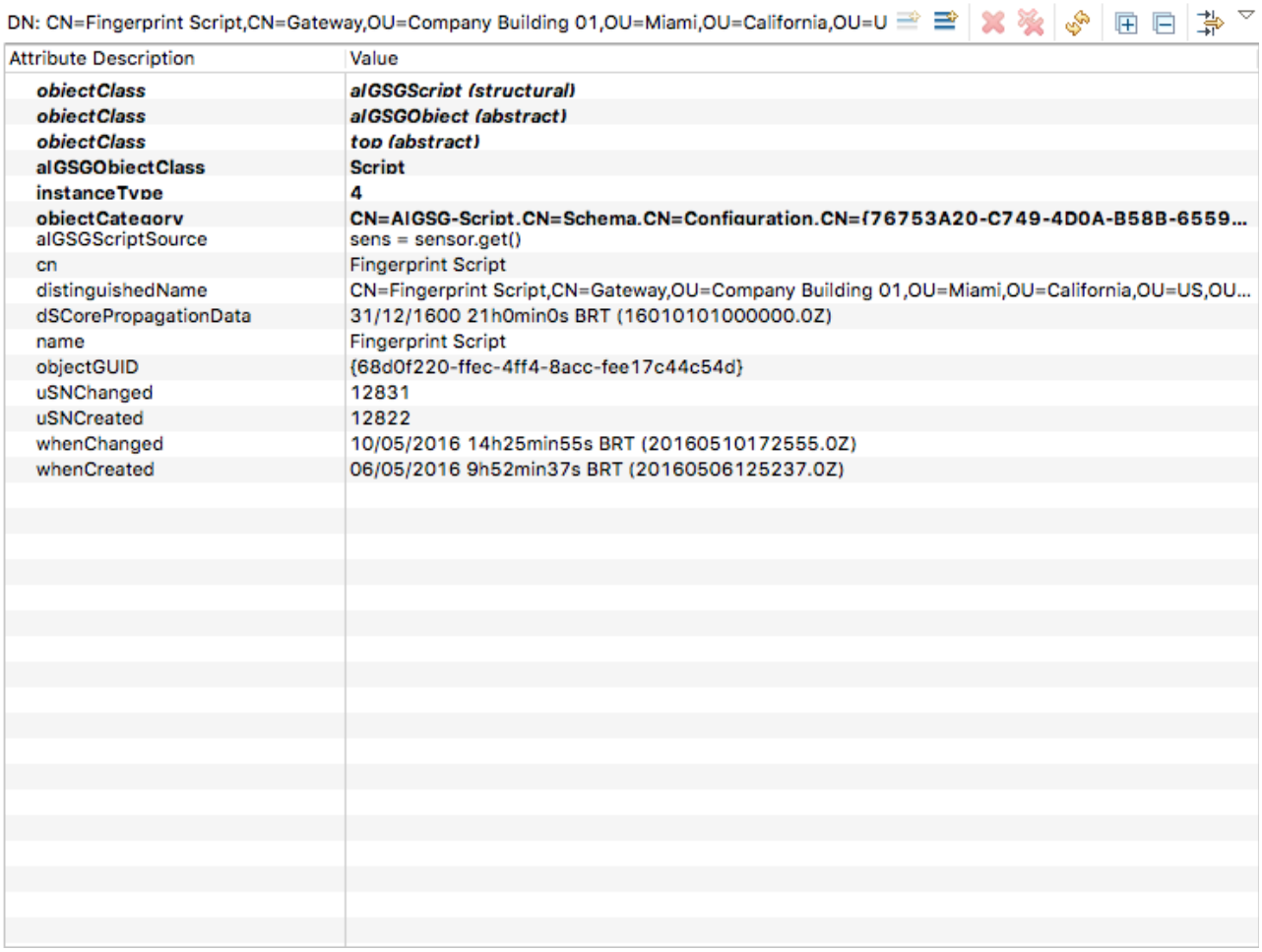
AigsgPolicyAllowEvent – attribute that allow the script to interact with sensor events.

AigsgPolicyAllowObject – attribute that allow the script to interact with sensor.

AigsgPolicyScript – attribute that associates the policy with a script.

2. Scripts inside LDS instance

The scripts inside LDS instance bring automation and inteligence to the IoT environment.
Using scripts, you automatize the system, creating rules of interaction between objects in LDS instance.


DN: CN=Fingerprint Script,CN=Gateway,OU=Company Building 01,OU=Miami,OU=California,OU=U

Attribute Description	Value
objectClass	***aIGSGScript (structural)***
objectClass	***aIGSGObject (abstract)***
objectClass	***top (abstract)***
aIGSGObjectClass	**Script**
instanceType	**4**
objectCategory	**CN=AIGSG-Script,CN=Schema,CN=Configuration,CN={76753A20-C749-4D0A-B58B-6559...**
aIGSGScriptSource	sens = sensor.get()
cn	Fingerprint Script
distinguishedName	CN=Fingerprint Script,CN=Gateway,OU=Company Building 01,OU=Miami,OU=California,OU=US,OU...
dSCorePropagationData	31/12/1600 21h0min0s BRT (16010101000000.0Z)
name	Fingerprint Script
objectGUID	{68d0f220-ffec-4ff4-8acc-fee17c44c54d}
uSNChanged	12831
uSNCreated	12822
whenChanged	10/05/2016 14h25min55s BRT (20160510172555.0Z)
whenCreated	06/05/2016 9h52min37s BRT (20160506125237.0Z)

Script inside LDS instance

The main attributes of a script object are:

AigsgScriptSource – attribute responsible to receive the LUA script that will automatize the IoT environment.

3. LUA script examples

ES uses LUA scripts inside LDS instance to automatize the IoT Enviroment.

Here are some script examples:

3.1. Lux sensor Turn ON or OFF a actuator

```
sens = sensor.get()
if (sens == nil) then
        return
end
value = sensor.get_value()
relay = object.get("path in LDAP to actuator.Relay")

ctx = script.context()

if (value =< 200.0) and (ctx.state == nil or ctx.state == false) then
        sensor.set_value(relay, false)
        ctx.state = true
        event.send(\{class = "test.ScriptEvent", msg = "Main lights turned OFF, turning
ON security lights"\})
elseif (value > 200.0) and (ctx.state == nil or ctx.state == true)  then
        sensor.set_value(relay, true)
        ctx.state = false
        event.send(\{class = "test.ScriptEvent", msg = "Main lights ON, turning OFF
security lights"\})
end}
```

In the script above, ES will receive values from a sensor, and if the values are equal or below 200.0, the actuator will be turned off. If the values received from the sensor are greater that 200.0, the actuator will be turned on.

This script is designed to receive values from a Lux Sensor and control a actuator in charge of Lights in a room.

3.2. Presence sensor turn ON or OFF a alarm actuator

```
sens = sensor.get()
if(sens == nil) then
	return
end

value = sensor.get_value()
relay = object.get("path in LDAP to actuator.Relay")

ctx = script.context()

if(value == true) then
	sensor.set_value(relay, true)
	ctx.state = true
	event.send({class = "test.ScriptEvent", msg = "Activate alarm"})

elseif(value == false) then
	sensor.set_value(relay, false)
	ctx.state = false
	event.send({class = "test.ScriptEvent", msg = "Deactivate alarm"})

end
```

In the script above, ES will receive values from a sensor, and if the values are equal 1, the actuator will be turned on. If the values received from the sensor are equal 0, the actuator will be turned off.

This script is designed to receive values from a Presence Sensor and control a actuator in charge of Alarm.

3.3. Temperature sensor turn ON or OFF a actuator

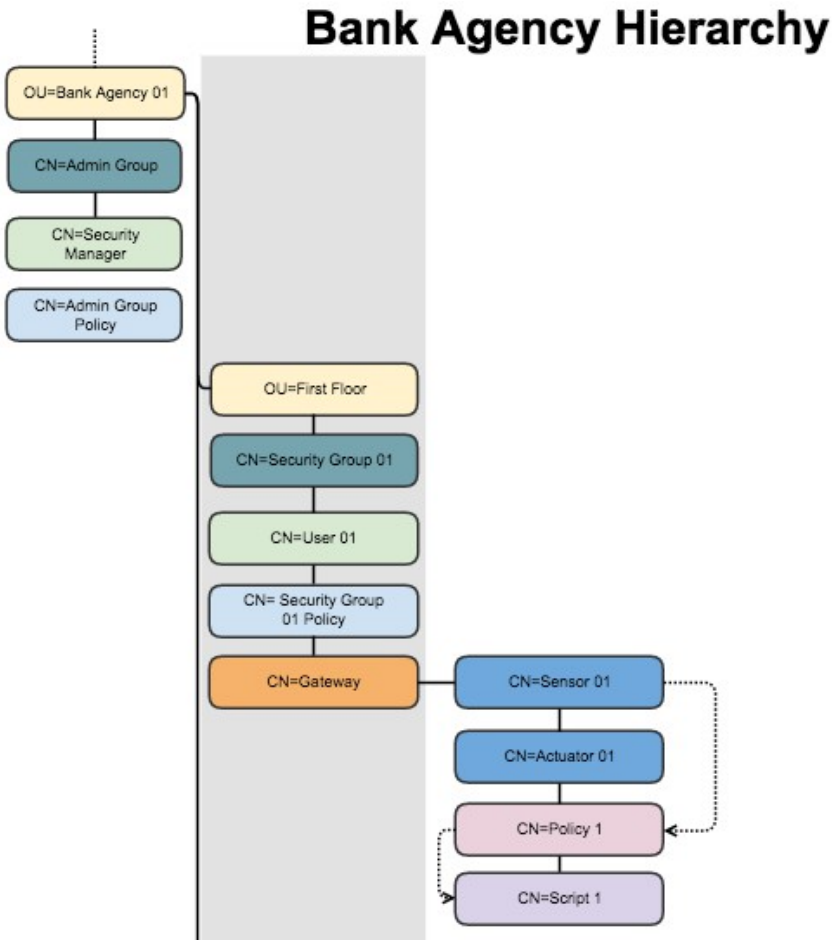
```
sens = sensor.get()
if (sens == nil) then
        return
end

value = sensor.get_value()
relay = object.get("path in LDAP to actuator.Relay")

ctx = script.context()

if (value >= 30.0) and (ctx.state == nil or ctx.state == false) then
        sensor.set_value(relay, false)
        ctx.state = true
        event.send({class = "test.ScriptEvent", msg = "High temperature! Turning FAN
ON"})

elseif (value <= 29.9) and (ctx.state == nil or ctx.state == true) then
        sensor.set_value(relay, true)
        ctx.state = false
        event.send({class = "test.ScriptEvent", msg = "Cooled down. Turning FAN OFF"})

end
```

In the script above, ES will receive values from a sensor, and if the values are equal or greater than 30, the actuator will be turned on. If the values received from the sensor are equal or below 29.9, the actuator will be turned off.

This script is designed to receive values from a Temperature Sensor and control a actuator in charge of a Air Conditioner or a FAN.

4. Objects association inside of LDS Instance

Here is a clear example of how the policies and scripts are associated inside the LDS Instance:



Bank sample structure in LDS Instance

In order to make the associations between objects in LDS Instance you must know that:

- A **sensor** point to a **policy** using the attribute **AigsgObjectPolicy;**
- A **policy** point to a **script** using the attribute **AigsgPolicyScript;**

Based on this, they relate as follow:

SENSOR > POLICY > SCRIPT

You can see that they relate in a one-way relation.

Sensor links to a policy, that calls a script, checking sensor values, and acting in case that it fills his needs.

5. Real life example script (Turn ON Air Conditioner)

Let`s imagine the following scenario:
A room with:

- Temperature Sensor
- Air Conditioner Actuator

In this scenario, the following script can be used:

```
### Temperature Sensor activates Air Conditioner ###
sens = sensor.get()
if (sens == nil) then
        return
end

value = sensor.get_value()
relay = object.get("path in LDAP to actuator.Relay")

ctx = script.context()

if (value >= 27.0) and (ctx.state == nil or ctx.state == false) then
        sensor.set_value(relay, false)
        ctx.state = true
        event.send({class = "test.ScriptEvent", msg = "High temperature! Turning FAN
ON"})

elseif (value <= 26.9) and (ctx.state == nil or ctx.state == true) then
        sensor.set_value(relay, true)
        ctx.state = false
        event.send({class = "test.ScriptEvent", msg = "Cooled down. Turning FAN OFF"})

end
```

Using this script in this scenario, we will have the 2 following situations:

1. When the Temperature inside the room is below 27 degrees Celsius, the Air Conditioner will be\turn OFF.

2. When the Temperature inside the room is higher than 27 degrees Celsius, the Air Conditioner will be\turn ON.

API (APPLICATION PROGRAMMING INTERFACE)

Event Server is provided with a series of API`s to use with it.

Here are the list of API`s:

libmclient is a communication library with ES server (https://github.com/AIGSG/libmclient)

es-proto - google protobuff description of the messages between es server and clients, all messages that can be send to es server are defined there (https://github.com/AIGSG/es-proto)

Python binding for libmclient (https://github.com/AIGSG/es-tools/tree/master/pylibm)

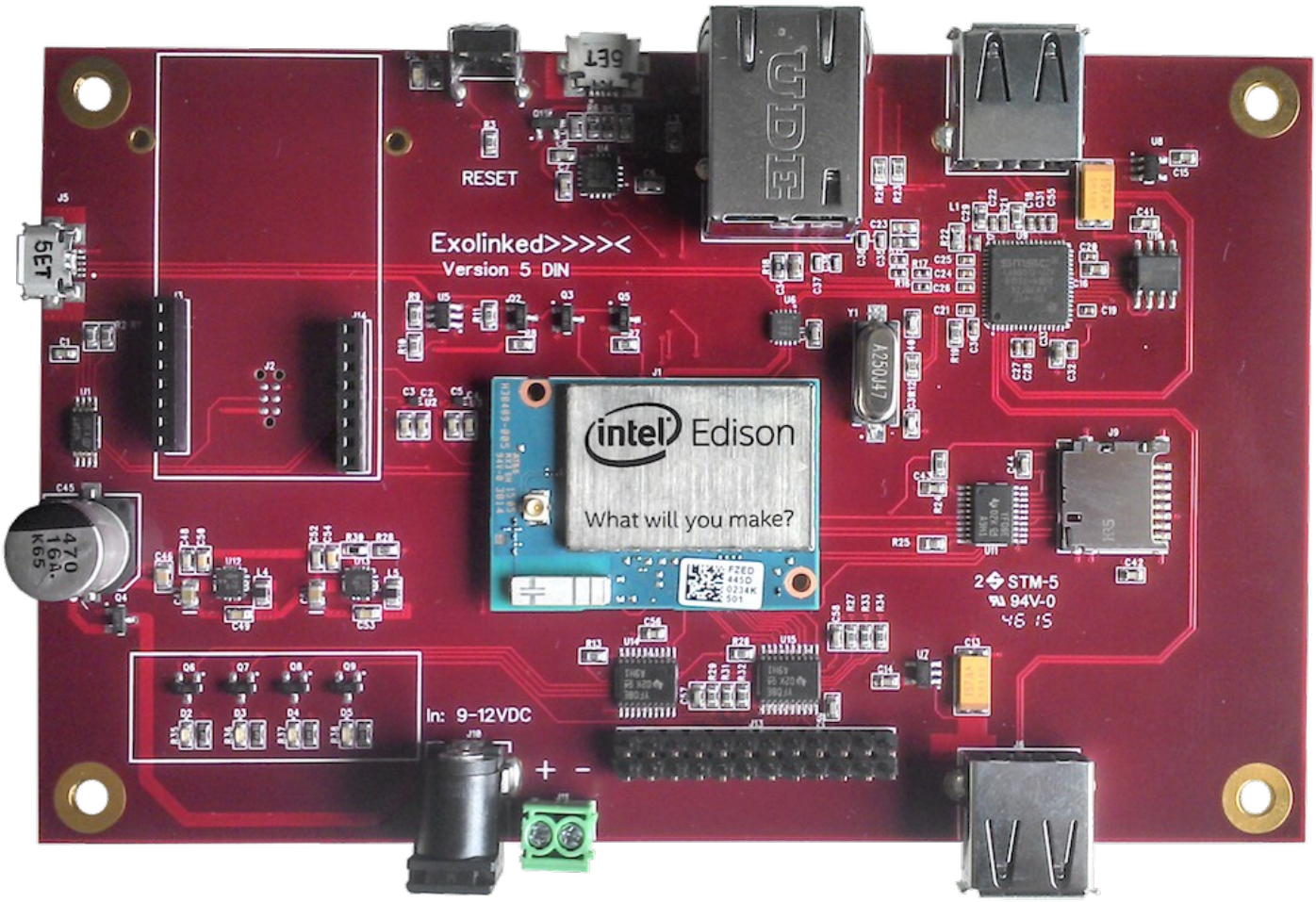
Tools developed using ES (https://github.com/AIGSG/es-tools)

libes is a C++ wrapper around libmclient and es-proto user api part (https://github.com/AIGSG/libes)

libdevapi - Designed to hide libmclient + es_proto + devapi interface and wrap everything as C++ (https://github.com/AIGSG/libdevapi)

AIGSG IoT Gateway (Powered by Intel® Edison)

AIGSG developed a IoT Gateway that is powered by Intel Edison processor.



AIGSG IoT Gateway

1. Flash Gateway using AIGSG Image

Windows
http://www.intel.com/content/www/us/en/support/boards-and-kits/000005795.html

Linux
http://www.intel.com/content/www/us/en/support/boards-and-kits/000005990.html

MAC OS X
http://www.intel.com/content/www/us/en/support/boards-and-kits/000005801.html

Follow the instructions detailed in Intel website, but use AIGSG Gateway Image instead of the image available in Intel website.

AIGSG Gateway Image has flashall.sh and flashall.bat.
flashall.sh – MAC OS X and Linux | flashall.bat – Windows

2. AIGSG Gateway basic info

Here are the access information of AIGSG Gateway:

2.1. IP Address:
Default usb0 IP address (USB-device through uUSB connector): 192.168.253.254

Default eth0 IP address (gateway_v1 or gateway_v2 motherboard): 192.168.254.254

2.2. Credentials
Username: root
Password: Aigsg2015Gateway

Image already comes with OpenVPN preinstalled. Configuring eth0 or wlan0 in /etc/network/interfaces to allow Internet access will cause Gateway to automatically connect to AIGSG VPN.

Image also already includes sensordaemon preconfigured with ES Server 10.20.20.16. Communicators can be connected either through USB Master or directly to Gateway through integrated communicator connector (/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyMFD1 are preconfigured).

Sensordaemon logs and configuration are located in /opt/sensordaemon/sensordaemon.

After connecting communicators, one can check /tmp/sd__dev_ttyXXXX files which should contain dump of detected communicators with connection type, hardware IDs, bus/radio addresses, serials and sensor/actuator types.

FIXME: sensordaemon needs to be restarted after changing sensors in Active Directory.

3. Setup Gateway to connect to Central ES

After flashing the Gateway, go to /var/lib/eventsrv folder and change eventsrv.conf file.

```
# Sample configuration file

# Database settings
Database:
  # Host name or address of the database server
  Host: localhost
  # Database name
  Name: eventsrv
  # Database user credentials
  User: eventsrv
  Password: eventsrv

# AMQP settings
Amqp:
  # Host name or address of the AMQP server
  ServerHost: <ES Primary Server IP>
  # Virtual host name
  VirtualHost: eventsrv
  # AMQP user credentials
  User: eventsrv
  Password: eventsrv

# Primary settings
Primary:
  # Path to installation directory
  BaseDirectory: /var/lib/eventsrv
  # Network address of this host
  HostAddress: <ES Primary Server IP>
  # Device serial number (device-local configuration only)
  SerialNumber: <Serial Number>
```

In Gateway configuration there`s no need to have LDAP configuration block. Only Primary Event Server connects to LDS Instance to retrieve objects.

Gateways only have access to it`s own structure inside LDS instance (Gateway object , Sensors, Actuators, Policies and Scripts inside of it.)