IoT with EPOS Tutorial

Table of contents

EPOS Hands-On

For the first contact with EPOS you can follow a step-by-step guide in EPOSMoteIII Quick-Start. If you are attending an on-site training, please, make sure you read this guide in advance in order to prepare the requirements. Bring to the class at least a Linux notebook with the GCC Toolchain for ARM installed and the SVN EPOS 2 for ARM code downloaded.

Guided Tutorial

The next sections will guide you to program an EPOSMoteIII for IoT solutions with SmartData and the UFSC's IoT platform, step-by-step. The guide is subdivided in tasks, each one explaining a feature of EPOS 2 important for interacting with the IoT infrastructure. The tasks are in the branch arm directory `app/tutorial`. As explained in Running EPOS on EPOSMoteIII the command `make APPLICATION=hello flash` tries to compile and write the application `app/hello.cc` with traits file `app/hello_traits.h` in the EPOSMoteIII via Serial/USB. We do not need to change traits files from the tasks in this tutorial, unless the instructor says otherwise. First of all, you can make a symbolic link from these two directories in the directory `app` with the following commands to
enable the makefile to find them.

```
cd app ln -s tutorial/* .
```

The SmartData abstraction used in EPOS 2 embeds the location of produced data that can be achieved automatically using trilateration and radio strength information. However, for simplicity of this tutorial, the instructors could ask you to compile EPOSMoteIII tutorial applications with fixed coordinates with the following commands replacing the values with the coordinates given by the instructors.

For tasks 1, 2 and 3 (for a standalone node, out of a Wireless Sensor Network):

```
make GXYZ="(0,1,2)" APPLICATION=app_name flash
```

For tasks 4 and 5 (for a network node, responding to a sink):

```
make XYZ="(0,1,2)" APPLICATION=app_name flash
```

Task 1 - My first SmartData with a local transducer

Here we show you how to interact with a sensor/actuator hardware with a SmartData using a Transducer abstraction in a standalone application. Your should complete the program `taks1_transducer.cc`. A transducer `must` define a method `sense` (for sensors) and a method `actuate` (actuators). This program has an example of a Transducer for a I2C thermometer Mediator, so you should complete to read the data from it. You could use `get` mediator method to read and assign the value to the SmartData.

```
data->_value = sensor.get();
```

You should also instantiate the SmartData passing as constructor arguments the numeric identifier of the device, the expiry time for the produced data and the mode. Note that the code uses a typedef to create a type that instantiate a SmartData with the temperature transducer created (My_Temperature). In this task we are going to create a local SmartData that is PRIVATE, which means that the data is not advertised in the network and that it does not receive commands from other nodes. This instantiation is exemplified below using the device number 0 and an expiry time of 15s.

```
My_Temperature t(0, 15000000, My_Temperature::PRIVATE);
```

You are also encouraged to use the `operator Value()`, and the `time()` and `location()` methods in the while loop, to print meta information of the SmartData.

Compile the code, program EPOSMoteIII and open the serial (with minicom or other tool) to see the output.

Task 2 - Observer attached to SmartData

Now we show how to create an Observer able to handle notifies from SmartData. The class Printer is an Observer that implements the method `update` which is called every time the observed SmartData triggers a `notify` (SmartData is an Observed). This notify is triggered every time the value of SmartData is updated so you do not need a while loop as in the previous task.

The program `task2_attach.cc` needs to be completed with the period (in microseconds) of the local SmartData for the transducer. This period indicates the frequency (1/period) that data will be read from the hardware. Complete it with a period of 5s. You should also complete it to print the SmartData every update as in the example below:

```
cout << "Temperature = " << (*_data) << " at " << _data->location() << ", " << _data->time() << endl;
```

Compile the code, program EPOSMoteIII and open the serial to see the output.

Task 3 - Sending SmartData to the UFSC IoT platform.

Here we show how to integrate a standalone EPOSMoteIII node to the IoT platform with the help of a Linux Gateway. This gateway can be any Linux device with USB interface and an Internet connection able to execute Python scripts. For this tutorial, we suggest using the same computer that you are connecting to the EPOSMoteIII with minicom. You will need to program EPOSMoteIII with the application `task3_iot_platform.cc`. This application prints SmartData with the Printer observer like the previous

application but now in a binary format to a USB port connected to the Linux gateway. The Linux gateway should execute the Python script `tools/eposiotgw/eposiotgw` that will read binary data received in the USB, interpret and send it to the IoT cloud. Execute `task3_iot_platform.cc` in the EPOSMoteIII, execute `eposiotgw` with the following options and the parameters given by the instructor:

```
./eposiotgw -j -D <domain> -U <user> -P <password> -g
```

The option `-g` indicates debug mode, which means that the script will only print data received from the USB port in the screen without sending it to the IoT cloud. After you are sure that the script is running properly execute the same command line without `-g`.

The `domain` parameter indicates a Cassandra database domain where your data will be stored in the cloud. The `user` and `password` indicates the credentials used to connect to the domain. The option `-j` tells the script to send data to the cloud in JSON. You can execute `./eposiotgw --help` to see other options from this command.

At this point, the gateway should be sending data to the cloud using JSON with a REST API and you can configure the Grafana dashboard panels from http://iot.lisha.ufsc.br/HomePage to visualize. Login with the credentials given by the instructors and click on the dashboard specified by them. Click on the name of the dashboard and after click in the option Edit. In this page, you should configure the credentials of the dashboard to fetch data from the same domain you are sending SmartData. You should also configure the `Interest`, that indicates which data will be shown in this dashboard. The Interest is an sphere with center in the coordinates `x`, `y`, `z` and radius `r`, for an device identifier `dev` and with a Unit. You should assure that the coordinates specified in your application SmartData is within the Interest sphere and using the same unit.

When you finish this configuration you should be able to visualize data produced by your sensor being forwarded by the Linux Gateway and presented in the Grafana dashboard.

Task 4 - Advertised SmartData

Here we are not using just standalone EPOSMoteIII connected via USB to a Linux gateway. We are using SmartData with a TSTP Wireless Sensor Network (WSN). Every EPOSMoteIII is a node from this network executing the application `task4_advertise.cc`. The instructors execute the sink of this network with the application `demo/demo_sink.cc`.

You should complete the application `task4_advertise.cc` to enable it to advertise SmartData to the sink as shown in the following code:

```
My_Temperature t(0, 15000000, My_Temperature::ADVERTISED);
```

The sink application `demo_sink.cc` has the same Printer observer from the previous application used to send data to the Linux gateway via USB. Now the sink is responsible for this operation for all messages advertised in the TSTP network.

Task 5 - Commanded SmartData

The last task is an example of commanded node. Here you should complete the application `task5_commanded.cc` to enable the EPOSMoteIII to receive commands from the sink to switch the LED on and off remotely.

```
```

```
Smart_Data<Switch_Sensor> my_led(0, 1000000, Smart_Data<Switch_Sensor>::COMMANDED);
```