SMART HOME SDK

Made by

Uri Reichman

Sensor:

In order to create a concrete sensor, you should inherit from Sensor (as seen in the picture).

```cpp
#ifndef SABATO_SENSOR4_HPP
#define SABATO_SENSOR4_HPP

#include "sensor.hpp"           // smart_home::Sensor, smart_home::AgentDetails
#include "mediator.hpp"         // smart_home::Mediator
#include "shared_ptr.hpp"       // experis::SharedPtr

namespace smart_home {

class Sensor4 : public Sensor {
public:
    // ~Sensor4();

    Sensor4(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details);

    virtual void Run();

};

} // smart_home

#endif  // SABATO_SENSOR4_HPP
```

Sensor4 is a concrete sensor

Sensor implementation goes as follows:

Event Publication:

Sensor gives you the ability to publish event threw the function

void Publish(const EventPtr& a_event);

EventPtr is SharedPtr<event> (made from EventType, Timestamp, location and payload).

When publishing event to this system, all the concrete controllers (will be explained later on) that subscribed to this type of event and from this sensor location, will receive the published event.

- You must publish a specific EventType – name, room and floor

```cpp
#include "sensor4.hpp"          // smart_home::Sensor4
#include "event.hpp"            // smart_home::Event
#include "shared_ptr.hpp"       // experis::SharedPtr
#include "definitions.hpp"      // SENSOR2_AND_3_TEST_SIZE, CONTROLLER1_EVENT_TYPE
#include <cstddef>              // size_t
#include <unistd.h>             // sleep

namespace smart_home {

smart_home::Agent::AgentPtr CreateAgent(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details) {
    return Agent::AgentPtr(new Sensor4(a_mediator, a_details));
}

Sensor4::Sensor4(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details)
: Sensor(a_mediator, a_details, true)
{
}

void Sensor4::Run() {
    Publish(EventPtr(new Event(EventType(CONTROLLER1_EVENT_TYPE, "2", "2"), Details().Position(), "sensor2_payload")));
    Publish(EventPtr(new Event(EventType(CONTROLLER1_EVENT_TYPE, "2", "2"), Details().Position(), "sensor2_payload")));
}

} // smart_home
```

Controller:

In order to create a concrete controller, you should inherit from Controller (as seen in the picture).

```
1 #ifndef SABATO_CONTROLLER1_HPP
2 #define SABATO_CONTROLLER1_HPP
3
4 #include "controller.hpp"       // smart_home::Controller, smart_home::AgentDetails
5 #include "event.hpp"            // smart_home::Event
6 #include "mediator.hpp"         // smart_home::Mediator
7 #include "shared_ptr.hpp"       // experis::SharedPtr
8
9 using namespace experis;
10
11 namespace smart_home {
12
13 class Controller1 : public Controller {
14 public:
15     // ~Controller1();
16
17     Controller1(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details);
18 };
19
20 } // smart_homeS
21
22 #endif  // SABATO_CONTROLLER1_HPP
```

Conctroller1 is a concrete controller

Controller implementation goes as follows:

Event Subscription:

Controller gives you the ability to subscribe to a specific event threw the function:

void Subscribe(const EventType& a_eventType, const HandlerPtr& a_handler);

EventType is the type of the event (made from name, room, and floor).
HandlerPtr is a SharedPtr<MultiThreadedHandler> of a concrete MultiThreadedHandler.

The concrete controller needs to implement a concrete MultiThreadedHandler for the event you want to subscribe.

- You must subscribe in the constructor (as shown in the picture)
- You can subscribe to all events, rooms and floors (strictly in this order)

Event Distribution:

In order to be a concrete MultiThreadedHandler, you should implement the function:

void Handle(const Controller::EventPtr& a_event);

The implementation of this function is code section where you receive event of the type you subscribed to.

```cpp
#include "controller1.hpp"      // smart_home::Controller1
#include "definitions.hpp"      // CONTROLLER1_EVENT_TYPE
#include <string>               // std::string
#include <assert.h>             // assert

namespace smart_home {

struct ControllerHandler : public MultiThreadedHandler {
    ControllerHandler(Controller1& a_this);

    virtual void Handle(const Controller::EventPtr& a_event);

private:
    Controller1& m_this;

};

ControllerHandler::ControllerHandler(Controller1& a_this)
: MultiThreadedHandler()
, m_this(a_this)
{
}

void ControllerHandler::Handle(const Controller::EventPtr& a_event) {
    assert(a_event->Type().Name() == CONTROLLER1_EVENT_TYPE);
    m_this.Log("test");
}

smart_home::Agent::AgentPtr CreateAgent(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details) {
    return Agent::AgentPtr(new Controller1(a_mediator, a_details));
}

Controller1::Controller1(const SharedPtr<Mediator>& a_mediator, const AgentDetails& a_details)
: Controller(a_mediator, a_details)
{
    Subscribe(EventType(CONTROLLER1_EVENT_TYPE, "2", "2"), HandlerPtr(new ControllerHandler(*this)));
}

} // smart_home
```

Configuration file:

The next picture is an example of a configuration file.
The file is made from "paragraphs" that represents an agent (sensor or controller).
The order of the line in each "paragraph" is not strict and you can change the order of the lines in each paragraph according to your will.

```
[Temprature-1-a]
type = ambient_temp
room = room_1_a
floor = 1
config = units: C; lower: -5; upper:55; period: 12

[hvac-a-1]
type = TestHVAC
room = room_1_a
floor = 1
log = hvac_log
config = iot:10.10.1.64; tmp:25; shutdown: Fire_Detected|ROOM_EMPTYS
```

Simulation:

After you implemented your concrete agent and the configuration file, you can start using the system.
You need to include central_hub.hpp in your application in order to use CentralHub.
The next picture is an example of that

We provide to the constructor of CentralHub a path to the configuration file.
After that, we start the system, getting the loggers for testing purposes, shutting down the system and taking logger message in order to validate the test.

```
CentralHub hub("./config.ini");

hub.Start();

loggers = hub.GetLoggers();


hub.Shutdown();

message = loggers["logger_name"]->Message();
```