With innovation in wireless sensing and communication technologies, we are now able to continuously monitor people's everyday conditions. Our goal is to design and develop a pervasive health software architecture on which developers can *add* new sensing devices, *com bine*
multiple information streams from the devices, and
develop
an innovative health application (e.g., Web or mobile). In this article, we showcase how to use HealthOS and to develop HealthOS applications and adapters.

With innovation in wireless sensing and communication technologies, we are now able to continuously monitor people's everyday conditions. Our goal is to design and develop a pervasive health software architecture on which developers can add new sensing devices, combine multiple information streams from the devices, and develop an innovative health application (e.g., Web or mobile). In this article, we showcase how to use HealthOS and to develop HealthOS applications and adapters.

* we define two consecutive right angle symbos (>>) to stand for command line.

0. Tested Environment

Operating System: Ubuntu Linux 12.04 LTS (> 10.x, HealthOS tested partially but not entirely) in Intel Core Duo architecture.

1. **Installation** (requred programs and libraries)

Python 2.7 >
Python SQAlchemy > 0.7.8
Python Tornado > 2.0 (http://www.tornadoweb.org/)
Python lxml > 2.4 (http://lxml.de/)
Python OAuth2 (https://github.com/simplegeo/python-oauth2)
Python SQLite (default in Python > 2.6)
OpenSSL (http://www.openssl.org/)

[SSL certificate setup] (please see the following articles).

1. www.flatmtn.com/article/setting-openssl-create-certificates
2. http://www.xenocafe.com/tutorials/linux/centos/openssl/self_signed_certificates/index.php

HealthOS: using SVN (Subversion), checkout the HealthOS code (SVN program download at http://subversion.apache.org/packages.html
)

>> svn checkout http://svn.hinrg.cs.jhu.edu/svn-private/healthos/

HealthOS directory structures: after downloading HealthOS you should see three

sub-directories under the main directory (healthos): cloud, client, and drivers

healthos

|__ cloud: directory where HealthOS server executables and source codes are contained
|__ client: directory where HealthOS client executables and source codes, as well as drivers, are contain
|__ drivers: directory where HealthOS Java drivers (developed to execute on the Android platform).

2. Configuration

: before running HealthOS system, you need to configure the following files to adapt HealthOS to your system environment.

[config.py]: cloud directory

BASEURL: set your machines IP address or DNS name. DB_NAME: set the name of database file to be used for SQLite. By default, name database.db is described.

[taskmanager.py]: cloud directory

PORT_REQ: the port number to which HealthOS applications connect. By default, it is 8082 PORT_INS: the port number through which HealthOS management service is provided. By default, it is 8083

Assuming BASEURL is lim.cs.jhu.edu and using default port numbers, you can connect HealthOS by typing https://lim.cs.jhu.edu:8082/lim (lim is user id of an imaginary user Lim). Similarly, HealthOS management service page can be access by typing https://lim.cs.jhu.edu:8083 in your web browser.

3. Running healthOS server

>> python taskmanager.py (cloud directory)

4. Running HealthOS client

>> python run_drivers.py (client directory)

This Python file is used to run the adapters developed in Python and the target environment for those adapters is PC environment. For mobile adapters, each adapters will be running individually.

5. Developing adapters

To expedite an adapter development, we provide a set of macro scripts. What do scripts do is to 1) create a generalized adapter, 2) register the adapter in HealthOS, and 3) setup a database tables and configurations for the newly created adapter to store collected data in HealthOS server.

1) Compose a JSON macro script

The JSON script describes a required information for an adapter to interface with the corresponding device. The JSON file needs to have the following fields:

packet: within this field, sub-fields are described to represent the sub-fields within a packet sent from a device. It has the four sub-fields: name, data type, size, state number. For data type, we currently support "int", "float", "string."

devicename: the name of device with which an adapter interfaces

devicetype: the type of device

drivername: an actual name of adapter

interface: communication interface: we currently have "socket," "Web," and "USB" (we currently support TinyOS serial modules).

[Example] JSON code for Zephyr BioHarness (zephyr.json) {

"packet": {
 "heartrate":[["heartrate", "uint", 2, 1]], "respiration": [["respiration", "float", 4, 2]],
"skintemp":[["skintemp", "float", 4, 3]],

"posture":[["posture", "uint", 2, 4]],

"pickaccel":[["pickaccel", "float", 4, 5]]

},

"devicename":"BioHarness",

"devicetype":"Physological Monitor",

"drivername":"bioharness_driver",

"interface":"Web"

}

There are separate five packets the Zephyr device transmits: heartrate, respiration, skintemp, posture, and pickaccel. And each packet define only a single field. For example, heartrate packet consists of a field called "
heartrate

" whose type of data is unsigned integer
uint
, of size
2
bytes, and state number
1
. If device sends a data in XML format, we describe "Web" for interface. Moreover, in XML format, the state number is meaningless. You can fill-in any integer value for this.

After composing this JSON script, put it under *macros* directory. >> mv zephyr.json macros

Then, run macro (developed in Linux/Unix **bash**)
>> ./macros/devinst.sh [json file] [database name] [interface type] [username] [device_id]
- [json file]: if not using Web interface, you need to compose a JSON file in 'macros' directory (within the directory there are lots of examples. - [database name]: a repository where all tables are stored assumed to be at HOSHOME. Just described the name
(e.g. put 'database.db' by default)

- [interface type]: type of interface (e.g. socket or TOSFTDI)

- [username]: HealthOS username

- [device_id]: unique identifier of the correspoding device

An example of running macro scripts may look like the following (using the same *zephyr.json* fi le).

./macro/devinst.sh zephyr.json database.db socket lim ZBH991615
assuming the device ID is "ZBH991615" and user name of the device owner is "lim". As the result of running the script, you will have default modules and configurations setup:

- **pipeline**: this is a combination of translator and db_connector. The combined module retrieves data from the database (*database.db*), and format in XML. Therefore, if you access *heartrate* data through https, " https://lim.cs.jhu.edu:8082/lim/BioHarness/ZBH991615/heartrate
"
99 2012-08-29 00:00:00
it will generate the above return value. As we described in JSON, it contains tag. The translator is put at *translator*, whereas db_connector being at *dbconnector* directory.

- **adapter**: default adapter is created.
- **device information**: new device information is added to database.

6. Developing your HealthOS applications using HealthOS REST APIs. HealthOS Java REST APIs are designed for Web/mobile applications to interact with HealthOS while abstracting out details of connecting HealthOS and parsing messages. Instead, the APIs serialize and deserialize the messages from HealthOS, thus developers can consume HealthOS data easily. For more detail on how to use it, please see the separate Webpage at https://sites.google.com/site/healthosapi/

7. Registering HealthOS applications using HealthOS store. In case your HealthOS applications are in the form of Web and store application data in HealthOS spaces, you can register your HealthOS applications to interact and save application data in HealthOS.

To do so, we first need to register you application through HealthOS. Go to the following URL:

https://BASEURL:[PORT_REQ]/apps
where BASEURL and PORT_REQ are the values we set in step 2 (*Configuration*). By selecting a name of your application (if there's no duplicate), you will be given a pair of keys (i.e., token and key for OAuth):

[**Connect to Your Application Space**] When you build your HealthOS application you could use the token and key as the third and forth parameters, respectively in
HealthOS
function in HealthOS Java APIs. See the following code:

```
HealthOS server = new HealthOS("https://lim.cs.jhu.edu", "medisn", "4UIeh8wXi3OsdQSG",
"Cgx8hBBQvWrOmayA");
```

[**Define Table and Insert Values**] To store your data, you first need to define a database table in your application space. To do so, we provide developers with a Java class called *TableRow*
.

```
TableRow patient = new TableRow("Patient");
```

Here, you can insert your columns in the table as follows:

patient.setProperty("name", "lim"); patient.setProperty("age", "25"); patient.setProperty("height", "175");
server.insert(patient);

The *insert* method store the given application data in HealthOS space.

[**Drop Table**] To drop the existing table, you can call *dropTable* method in HealthOS class. For example, assuming the HealthOS class generate its object called server:

server.dropTable("Patient")

[**Querying Data**]
To query data in your application's space in HealthOS, you can use *Query* class. In doing so, you can add *filters* by calling a method called *addFilter. The following is an example:*

Query q = new Query("Patient"); // argument for table name q.addFilter("name", Query.*EQUAL*, "test");

q.addFilter("salary", Query.EQUAL, 1234);

After constructing this query object, you pass it to HealthOS by calling prepare method with

the argument for query object:

```
result = server.prepare(q);
```

The *result* contains the XMLs matching the query condition.