DIY Zoning: xAP Protocol Support

Table of contents

1. Introduction

Originally, xAP protocol support wasn't planned. However, the demand for it was surfacing time to time, so I guess DZ will support xAP.

So, let's roll.

Note:
Information presented on this page relates to work in progress and is subject to change and/or differ from implementation details until further notice.

2. Data Representation

Normally, the sensor data will be represented by a floating point value - it is the responsibility of the consumer to normalize the data for display.

In addition to normal data values, there will be two special cases:

- **Unknown:** represented by a literal `U`. This value may appear for short periods of time after the system startup, and should be treated as transient.
- **Failure:** represented by a literal `F`, followed by a space and then a human-readable failure description, if available.

3. DAC

It seems that DAC xAP support will be pretty simple: xAP broadcaster will pretend to be a logging device. Only sensor data will be exposed via xAP broadcaster.

xAP support for DAC will not expose any controls. The first and foremost reason for this is that such support will break the abstractions inherent to DZ, thus making the system fragile. The secondary reason is that xAP doesn't provide any support for security and authentication.

3.1. Message Header

```
xap-header
{
v=12
hop=1
```

```
uid=FF????00
class=dz.dac
source=DZ.DAC.<host-name>
}
```

<*host-name*> is a string obtained with `InetAddress.getLocalHost()`, with everything after the first dot discarded.

3.2. Message Body

The rest of xAP message will consist of sensor data packets. There will be three kinds of data blocks:

```
dz.dac.temperature
{
timestamp=yyyy-MM-dd'T'HH:mm:ss.SSSZ
sensor.address=T<1-wire-address>
sensor.temperature=<temperature-centigree>
}

dz.dac.relative-humidity
{
timestamp=yyyy-MM-dd'T'HH:mm:ss.SSSZ
sensor.address=H<1-wire-address>
sensor.humidity=<relative-humidity-percent>
}

dz.dac.pressure
{
timestamp=yyyy-MM-dd'T'HH:mm:ss.SSSZ
sensor.address=P<1-wire-address>
sensor.pressure=<pressure-mbar>
}
```

At this point, the data from all sensors will be sent out at the same time. However, this brings up a concern about the maximum allowed UDP packet size (depending on MTU settings, about 1500 bytes), so it is possible that in the future the notifications will be either fragmented, or sent out in blocks of one or more.

Warning:

When listening to notifications, do **not** assume that you will get all the data in a single packet.

4. CORE as a producer

To an outside observer, CORE will look like a single entity containing some thermostats. This is questionable (if you don't like it, **now** is a good time to tell me about it), but logical

from my prospective. If it breaks compatibility with any xAP products, I'd like to know about it - it's easily fixable, at least at this stage.

4.1. Message Header

```
xap-header
{
v=12
hop=1
uid=FF????00
class=dz.core
source=DZ.CORE.<host-name>
}
```

<*host-name*> is a string obtained with `InetAddress.getLocalHost()`, with everything after the first dot discarded.

4.2. Message Body

The message body represents a thermostat status. A typical body will look like this:

```
dz.core.thermostat.status
{
timestamp=yyyy-MM-dd'T'HH:mm:ss.SSSZ
thermostat.name=Storage Room
thermostat.setpoint=26.0
thermostat.temperature=31.375
thermostat.voting=true
thermostat.hold=false
thermostat.disabled=false
}
```

Note:
Keep in mind that the temperature is expressed in degrees Celcius.

4.3. Unanswered Questions

How do I represent a HVAC mode (heating/cooling/off/etc.)? This is not a per-thermostat setting, but per-unit setting. There may be more than one unit that DZ controls. It is not clear how do I expose the units, and whether they should be exposed at all. It is also possible that at some point the mode *will* become a per-thermostat setting.

5. CORE as a consumer

To be continued...