DIY Zoning: xPL Protocol Support

Table of contents

1. Introduction

xPL protocol support is a followup to xAP protocol support. xAP and xPL have a lot in common, so it a decision to add xPL support was a no-brainer.

Note:

Whenever a piece of information about xPL support seems to be missing, it probably can be found in the section describing xAP protocol support. It is strongly recommended to get familiar with xAP support implementation details before trying to understand details of xPL support.

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

xPL data representation follows the same guidelines as xAP data representation.

3. DAC

The only difference from DAC xAP support is the packet format.

DAC implements the xPL's SENSOR.BASIC message schema. Notifications (at this point) are sent using `XPL-STAT`, due to the way xPL driver is implemented. It is possible that in the future they will support `XPL-TRIG` instead. Under no circumstances DAC will send out `HBEAT` packets.

3.1. Message Header

```
xpl-stat
{
hop=1
source=DZ.DAC.<host-name>
target=*
}
```

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

3.2. Message Body

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

```
sensor.basic
{
timestamp=yyyy-MM-dd'T'HH:mm:ss.SSSZ
device=T<1-wire-address>
type=temp
current=<temperature-centigree>
}

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

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

Note that `relative-humidity` and `pressure` sensor types are not defined by SENSOR.BASIC specification - hopefully, they will be added.

Same applies to the `timestamp` entry - it is not included into the current version of `SENSOR.BASIC` specification. However, I believe that it makes perfect sense to include it (at least as an optional entry) to improve quality of service for time critical systems.

In particular, DZ employs PID controllers to calculate control signal values, and few seconds delay will seriously degrade the control signal quality, unless the delay is accounted for - by including the timestamp into the data sample packet. Even though the control signal value will be distorted during the blackout, it will become correct as soon as all the delayed data is delivered to the PID controller.

Note:

Delays of tens of seconds are not unusual even for quite simple networks, especially if there are wireless components involved. Even UDP, which is a non-guaranteed delivery protocol, can queue packets and them deliver them all at once.

4. CORE as a producer

Unlike xAP, xPL supports the concept of a *trigger* (or, I wasn't able to find it in xAP docs - correct me if I'm wrong). Consequently, xPL messages issued by CORE *may* have more variety than xAP messages. All of them will be `XPL-TRIG` messages.

For initial implementation, however, there will be only one message that encapsulates all the information. To keep it simple. The downside is that even though the message will be a trigger, some of data elements will not change their value across several trigger messages, thus forcing the consumer to keep track of older values and detect the changes themselves.

4.1. Message Header

```
xpl-trig
{
hop=1
source=DZ.CORE.<host-name>
target=*
}
```

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

4.2. Message Body

xPL documentation doesn't seem to have any reference to thermostats, so we'll have to introduce our own message schema. Let's call it `DZ.CORE`.

A typical message body will look like this:

```
dz.core
{
timestamp=2005-07-23T16:40:12.374-0700
zone=Storage Room
setpoint=26.0
temp=31.375
voting=true
hold=false
disabled=false
}
```

Note:

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

5. CORE as a consumer

It doesn't seem that CONTROL.BASIC will be sufficient. Most probably, CORE will accept the same message it sends out as a producer, the only difference being `XML-CMND` instead of `XPL-TRIG`, and the target matching the string produced by an instance as a source. Of course, `zone` and `temp` values can't be set :)

To be continued...