Embedded Engineer Application Assignment

File name	Assessment-EmbeddedEngineer-A3.docx
Author	Daniel Reus / Francesc Subirada
Version	A3
Date	October 2015
Category	Application Assignment

Introduction

As a part of our application procedure we perform a technical assessment. This assessment consists of two major components:

1. We ask you, the applicant, to fulfill a technical assignment
2. After fulfilling the assignment you showcase the functional product and there is a technical discussion between you and relevant team members about how you fulfilled the assignment.

While doing the assignment, here are some things to keep in mind:

- It could happen that there's not enough time to implement all the requested functionality. However we do want to see something that works, even if it doesn't have all the requested functionality. Make sure you facilitate this for yourself.
- The requirements for the product are 'hidden' in the description text. Some are more essential than others. It's your job to prioritize.
- We will look at several aspects of your implementation. This includes (but is not limited to):
    - Code style
    - Code quality
    - Code Documentation
    - Code Efficiency
    - Memory leaks
    - Version Control
- You have to use Open Source Tools & Frameworks. We suggest Eclipse CDT plus GNU gcc, you can change it but you'll be requested why.
- The projects have to be in GitHub and use TDD approach.
- We would like a UML like design approach for the system.
- The implementation should be in ANSI C and be able to be ported to an embedded platform but we want the implementation be able to run as a Linux executable (as the one to be integrated in a CI environment) and so you can mock required HW and services.

The Assignment

We have some systems that gather samples from all kinds of measurements and report them to a service. There are other systems that request the current value for some of these parameters from this service. Your assignment is to implement this service. It is to be implemented in the C language. We suggest using a library called ezxml to handle the XML parts of the

assignment. You are free to modify it where needed. This service has 2 implementation platforms:

1) An Embedded IoT platform for the Client
    a. Battery Operated
    b. Connects every 15 min to the server
    c. Uses an MCU ((we suggest a ARM M0+ one, you can change it but you'll be requested why) with 32KB RAM and 256KB Flash.

2) An Linux Platform for the Server

Detailed description

The measuring clients report these measurements by sending a XML messages over TCP to port 6423 of some registry service .The xml messages consist of a root tag called 'update', inside it there is any number of (unique) keys as tags, with values as text content. For example:

```
<update>
      <personsPassed>12</personsPassed>
      <lightCondition>bright</lightCondition>
      <humidity>52</humidiy>
</update>
```

Once the payload is delivered the client can either send another update or disconnect.
On the other side of this service there will be several systems that request the current status for either some specific values, or for all of the currently known values. They connect to the service in the same way as the measurement devices. When connected they send an XML message with a root tag called 'retrieve' having 0..x child tags called 'key' and the name of the desired parameter as character content. When it specifies no parameters the server returns all the known parameters. For example:

```
<retrieve/>
```

To retrieve all currently known values, or

```
<retrieve>
      <key>personsPassed</key>
      <key>humidity</key>
</retrieve>
```

to retrieve the current value for the 'personsPassed' and 'humidity' parameters. The server should return an XML message with parent tag 'status' and a child tag per requested parameter having the name of the parameter as tag name and the value of the parameter as character value. Like so:

```
<status>
      <personsPassed>12</personsPassed>
      <humidity>52</humidiy>
</status>
```

After receiving this update the client can either request more parameters or close the connection.
The output XML of the service should be 'prettyfied', similar to the examples (indent child nodes), so that when a human 'client' queries the system the output is easy to read. When the service is shut-down and re-started it should start up with the last known state for all the parameters. When a client sends an invalid request the service should return nothing and close the connection.