Temperature logger synchronization

Introduction

Two or more installations of the temperature logger software can be configured to synchronize incoming and stored measurements. With this synchronization software one can inspect measurements on one site, including data from sensors only reachable from another site. The synchronization is available as part of the temperature logger software version 1.9.8.
Many topologies can be configured using two or more installations of this software. Each installation can act as a measurement data source, and/or it can extract data from one or more data sources. Incoming measurements, from a base station, a network station and/or from another PC, are propagated as source data to other installations. The drawing below depicts a basic configuration:

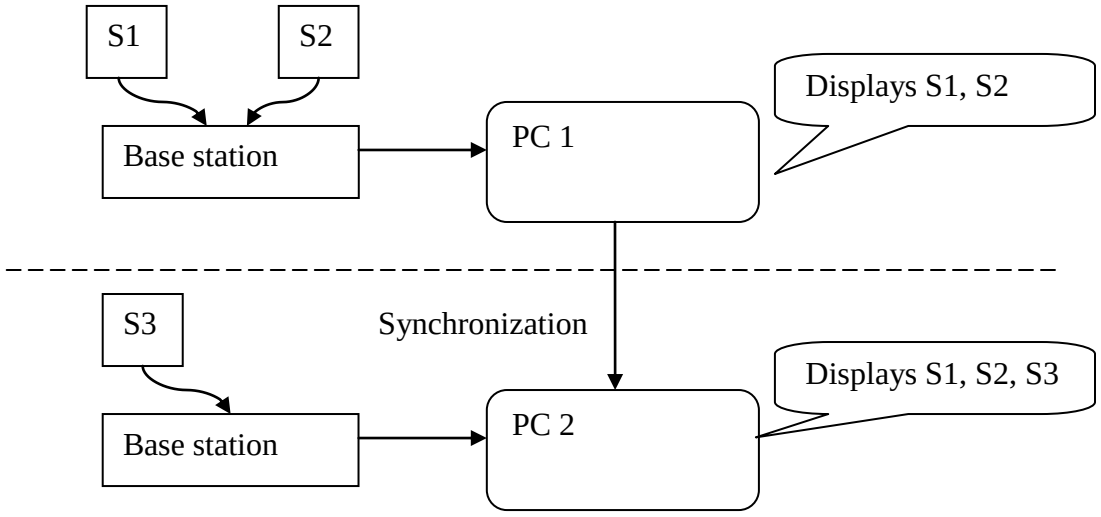

Figure 1: *A basic synchronization of two temperature logger software installations: On site 1 two sensors S1 and S2 are processed by computer PC1. A second computer PC2 on site 2, combines incoming measurements of sensor S3 with data from PC1, and shows data from all 3 sensors.*

Implementation

The synchronization runs over an HTTP protocol using TCP/IP, and uses user defined network ports when specified. Each software installation needs a configuration file (sync.xml, located in the installation directory) that defines the incoming and outgoing network nodes. No user interaction is required. The configuration file is an xml file that can be edited by a normal text editor or a specialized xml editor. An example file is enclosed in the temperature logger software. The sync.xml configuration file contains the following:

- 0 or 1 **httpserver** entities: Specifies details for the HTTP server that is used to listen to incoming data requests.

- 0 or 1 **postserver** entities: This element specifies a list of TCP/IP addresses & ports to post new update information to.

The basic example above would use the following two configuration files:

The configuration file for pc1 specifies the address that is used to push new data to pc2. When needed, pc2 will ask for additional (historical) data. In order to let the pc1 handle and provide these requests, the httpserver entry has to be configured to allow handling the incoming data requests.

```
<?xml version="1.0" encoding="utf-8"?>
<sync>
<!-- pc1  -->
  <httpserver listen="49160">
  <source name="pc2.lan"
          callbackport="49161"
          maxcallbackdays="14">
  </source>
  </httpserver>
  <postserver>
      <post url="pc2.lan:49161" maxcallbackdays="14">
      </post>
  </postserver>
</sync>
```

Figure 2: *The sync.xml configuration file for pc1 of the example.*

The configuration file for pc2 specifies that it can accept requests from pc1.lan. It will send callbacks (requests for more data) to this pc1 on the callbackport.

```
<?xml version="1.0" encoding="utf-8"?>
<sync>
<!-- pc2  -->
   <httpserver listen="49161">
    <source name="pc1.lan"
            callbackport="49160">
      </source>
  </httpserver>
</sync>
```

Figure 3: The *sync.xml configuration file for pc2 of the example.*

Note the port numbers should match the respective connections.

Each configuration is set up during the start of the temperature logger service. The temperature logger service will watch the sync.xml file and will update its synchronization configuration as soon as the sync.xml has changed. Therefore, a synchronization change takes effect by changing the sync.xml file.

The source and post elements can both enclose the 'maxcallbackdays' attribute. This attribute specifies the amount of days back from current time the synchronization is allowed to call for more data. If not set, the entire dataset is requested. With this attribute the amount of data requests can be limited; also the amount of time it takes to synchronize can be limited by reducing the call back period.

The source element has the attribute 'sequence', which can be set to "yes". This indicates the server to synchronize its callback requests to the source such that requests are made in sequence, and not parallel. This is useful for sources with limited capacity like the LAN base stations.

LAN base station.

A LAN base station can be a source in the synchronization network. This is done by providing the base station with a messenger rule that will post data to a temperature logger server. The URL that defines the destination of the message should specify the used port as well, using the colon notation. The rule should be of an HTTP post request type, and have the following message body:

```
type=$q&id=$i&time=$S&v=$v&rssi=$r&missing=$w
```

This message body specifies the sensor id and type, the timestamp of the measurement (number of seconds after 1-1-2000 UTC), the measurement value, the rssi value and finally the missing parameter specifying the last time a successful update was sent to the server. Note that these parameters are required for proper functioning. The server will process these messages and merge them into its dataset. It then will callback the base station with requests for missing data - if necessary. The missing data is provided as xml data on the page data.xml of the lan base stations webserver. The callback port for the LAN base station is 80 (default HTTP port).

Below the current example is extended by adding 2 LAN base stations. See the user manual of the BS1000 for more details on how to set the messenger rules.

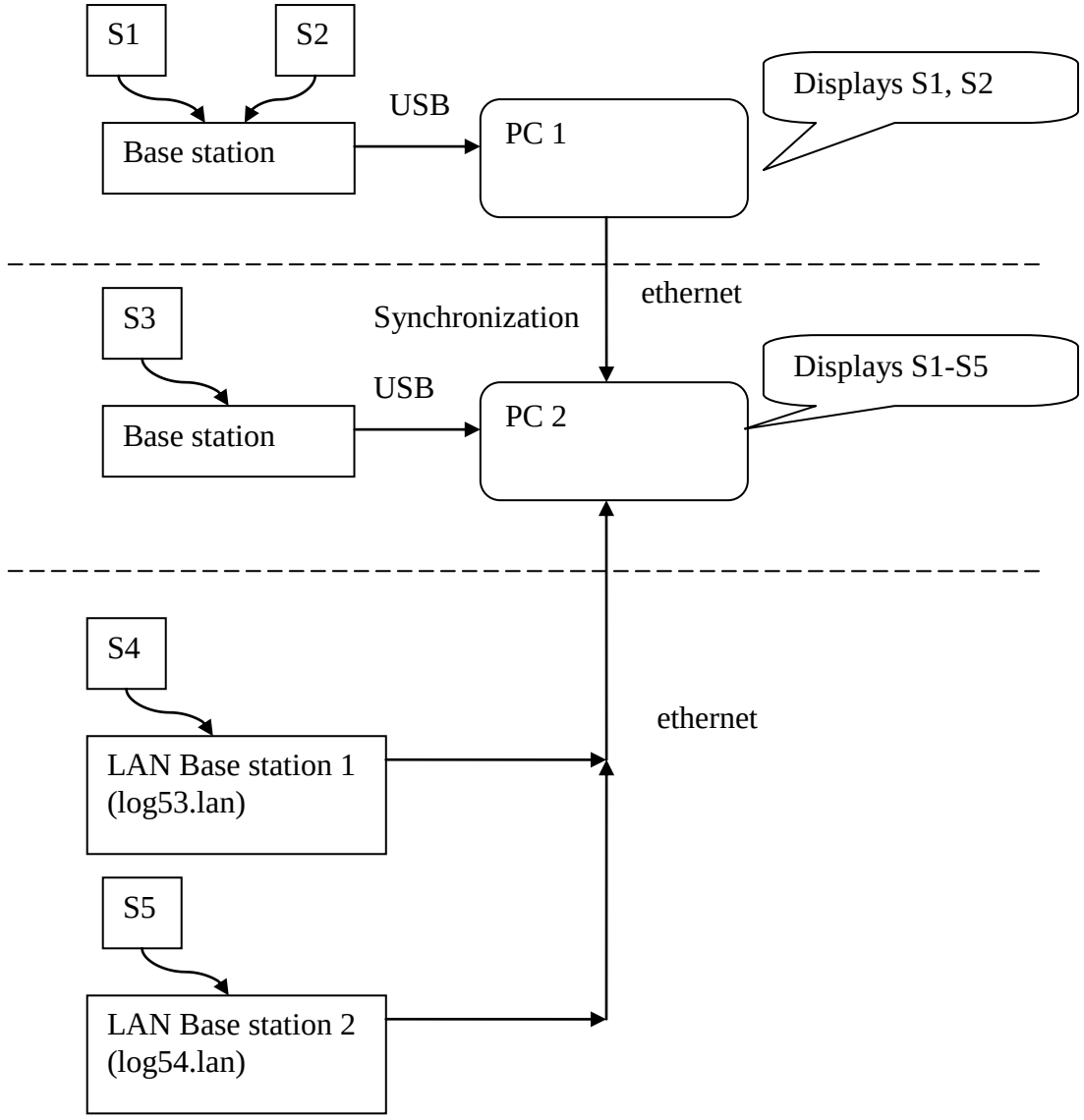

Figure 4*: The basic synchronization of two temperature logger software installations extended with two LAN base stations. PC 2 combines data from sensor 1 to 5.*

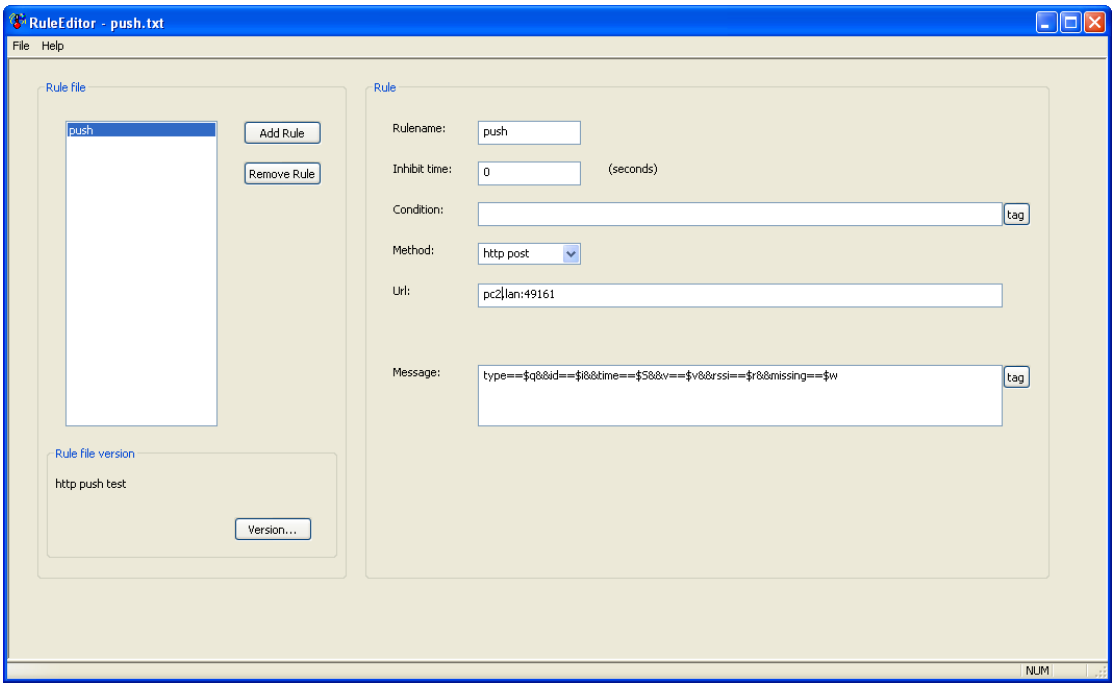

Figure 5: *With the rule editor tool it is possible to create a messenger rule file for the BS1000 LAN base station. This file is uploaded to the lan station on its webserver.*

Figure 6: *On this page of the BS1000 webserver you can upload the messenger rule file that defines the messenger rules to be applied on incoming measurements.*

```
<?xml version="1.0" encoding="utf-8"?>
<sync>
<!-- pc2  -->
   <httpserver listen="49161">
    <source name="pc1.lan"
            callbackport="49160">
      </source>
    <source name="log54.lan"
            callbackport="80"
            sequence="yes"
            maxcallbackdays="7">
      </source>
    <source name="log55.lan"
            callbackport="80"
            sequence="yes"
            maxcallbackdays="7">
      </source>
  </httpserver>
</sync>
```

Figure 7: *In the example system of figure 2 the above configuration file is used.Here log54 and log55 are added as source. The sequence attributes is set to "yes" in order to prevent an overrun in requests to these webservers.*

Monitor the synchronization

The temperature logger software shows the number of in- and outgoing measurements per minute on a separate synchronization window. Per peer node it shows how many measurements were received and how many were posted per minute. The screen is updated every 15 seconds. Also it shows the last time of measurement that was received. This gives a rough indication on how well synchronization is performed.

Time

At this moment, it is assumed the system clocks of all nodes in the synchronization network are running within 15 seconds precision. Usually this can be accomplished by letting the system set its time by a time server. The LAN base stations are capable of synchronizing to a time server as well.

Background

This paragraph gives some background on how the synchronization is implemented. The update messages are implemented as follows:

1. A sensor sends in new data
2. The postserver will broadcast the new data to all post clients

If a client fails in step 2, the time of this measurement is marked as missing for that specific client.
In that case the next steps will occur:

3. As soon as new data is received and processed again, an HTTP request is sent by the client back to the peer on the callback port with a request for more data.
4. The data is composed and sent over. The data may not be complete. However, it is consecutive, in order and it is the earliest chunk possible.

5. The data is received and merged into the local system.
6. The period of received data is acknowledged to the peer.
7. The peer uses this acknowledge to update the missing time parameter.
8. These steps are repeated until all data is acknowledged.