Technical Documentation for the control code of the Arduino system in UXC at CMS

The purpose of this document is to overview and analyse the main principles of the shell script programmed by Nikolaos Chrysogelos which is responsible for the control and the proper function of the temperature and dew-point sensors in CMS cavern which are governed by an Arduino system. The operations, the main functionalities and the implementation of the script are demonstrated and explained, while the limitations and how it should be used are noted on the following pages.

Nikolaos Chrysogelos
nkchrys@gmail.com

Francesco Palmonari
Francesco.Palmonari@cern.ch

Contents

1.1 Overview

A simple sketch of the architecture of the Arduino system with the sensors is given on this section. Approximately 500 temperature and dew-point sensors were prepared, calibrated and installed by Francesco Palmonari and Nikolaos Chrysogelos for measurements targeted to online monitoring at critical areas of the CMS detector. The network of sensors is read out, and controlled by 10 Arduino AT Mega 2560 installed in the FAR and NEAR side of the CMS cavern. Each Arduino is publishing its data in a dedicated html web page from where it is possible to read the values of each sensor.

In order to control and check the proper functionality of each Arduino device as well as the correctness of the measurements of each sensor a control script was created by Nikolaos Chrysogelos and its analysis is the scope of this document.

The following figure shows a simplified picture of what was installed in CMS cavern. In total there are ten Arduinos. Each one of them has a unique web page where it publishes the results (10.176.2.142 /143/…/152). The devices may have up to 5 one wire buses according to our needs. On each cable, there are 10 temperature and 5 dew-point sensors. On this point it is important to note that the flexibility of this system allows us not only to connect more Arduinos but also to expand the network of sensors very easily by attaching sensors on the one wire bus wherever we need measurements. This means that the number of sensors and arduinos which were presented above, in fact it is much larger and is modified every day. But the purpose of this document is not to present an accurate presentation of the hardware which was used for the specific application but to offer a brief summary of what was installed in order to reveal the need for controlling the devices and the sensors with the help of a script. More details on the architecture of the system may be found here.

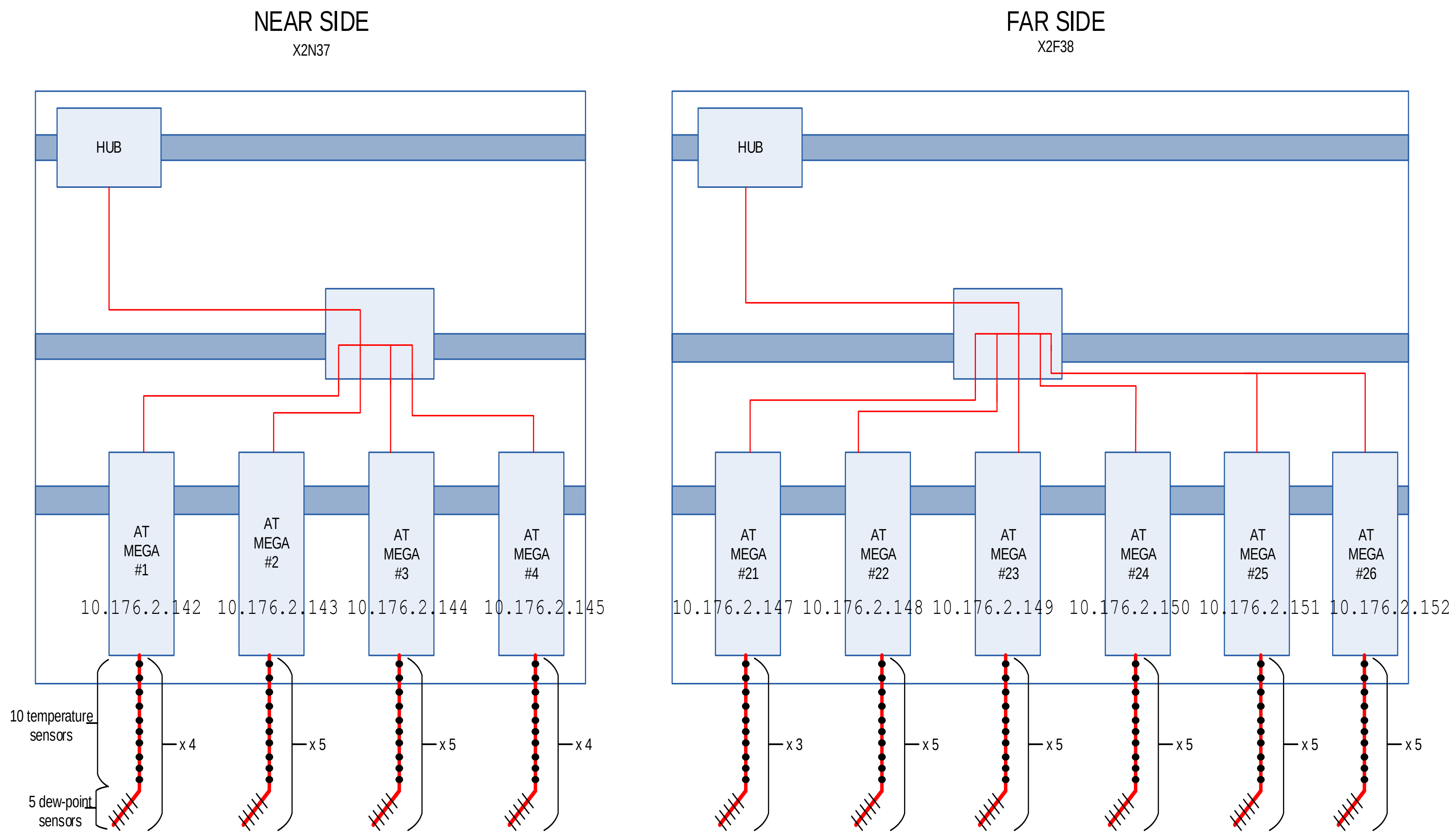


Figure 1. Architecture sketch of the Arduino system.

1.2 Implementation

It is a shell script that also used the awk text processing language. The main functionalities of the control script are the following;

- Check for **connection problems** of the Arduino in case it is not powered on or for some reason it is not connected to the network.

- Check for **missing sensors** in case they were damaged. (If they were removed manually the script should run from the beginning in order to save again the initial IDs of the sensors)

- Check for **non updated sensors** in case they are not updating their values anymore because of problem.

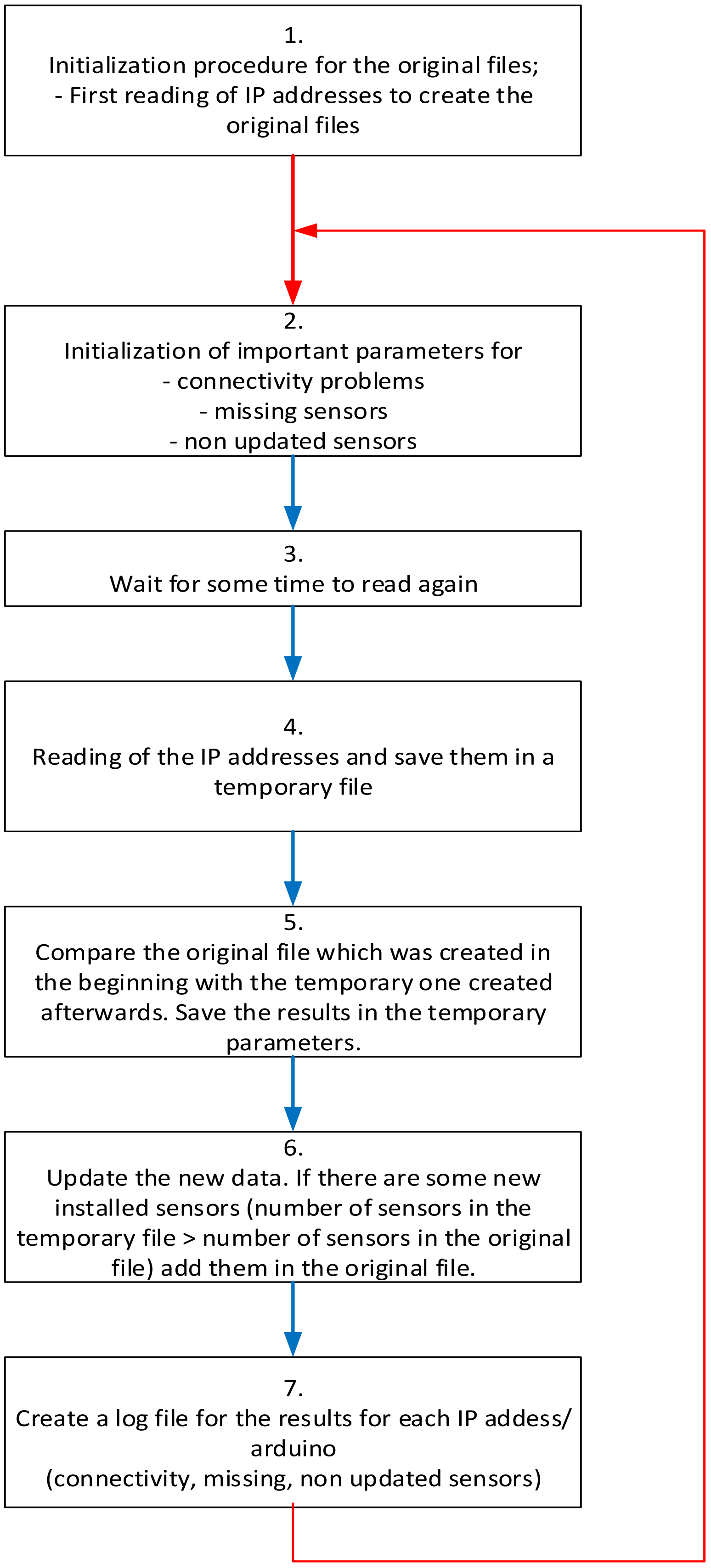
- **Detection and addition of new sensors** on the control loop of the script which were connected afterwards in our system.

- **Logging the results of the check,** after the script has finished its control, the results are saved on the log_file.

All of these are done by comparing two files, the original and the temporary one. In the beginning of the script we save the initial IDs of the sensors for each of the ten Arduino devices on the corresponding original file (original_142 – original_152). On this point it is noted that a specific IP address is assigned to each Arduino, that's why we may name each Arduino device by its dedicated IP address. The creation of the original files is done only once at the start of the execution of the code. That's why if a sensor is removed from the network, the original files should be deleted and run the script from the beginning.

After the initializations the script enters to a loop which is repeated over a specific period of time. For each loop, a temporary file is created for each Arduino (check_142 - check_152) and it is compared with the original one to check for malfunctions. After the comparison, the results for possible problems are displayed and logged in the log_file. Finally, if the number of sensors which are in the original file is smaller than the number in the temporary one which means that new sensors were connected in our system, then they will be added in the original file without any manual interruption by the user.

The operation of the script code is summarized on the flow diagram in the next page.


1.
Initialization procedure for the original files;
- First reading of IP addresses to create the original files
2.
Initialization of important parameters for
- connectivity problems
- missing sensors
- non updated sensors
3.
Wait for some time to read again
4.
Reading of the IP addresses and save them in a temporary file
5.
Compare the original file which was created in the beginning with the temporary one created afterwards. Save the results in the temporary parameters.
6.
Update the new data. If there are some new installed sensors (number of sensors in the temporary file > number of sensors in the original file) add them in the original file.
7.
Create a log file for the results for each IP addess/ arduino
(connectivity, missing, non updated sensors)

On the flow diagram the main core of the program was presented. On the following of this section the code of the script along with comments is shown for each box in the previous figure.

1. Initialization,
For each IP address we run a loop which saves their data. If the original file already exists, we exit the function. Otherwise we call the wget function to retrieve the data. If its output is different from 0, it means that there is an error with the connection. Wget saves the data in a file named 142/143/…/152 which afterwards we edit in order to keep only the **time** and the **ID** of the sensor on the original_142/…/152 which are the only useful values for the comparison with the temporary file.

```
#first reading of sensors
echo "First Reading of IP Addresses"

#create the initial file for each IP address
edit $"original_"


#edit function: creates the initial original file for all the
arduinos/IP_addresses
edit(){
     for i in 2 3 4 5 7 8 9 10 11 12
     do
#the name of the original file is the same with the IP_address
          let name=140+$i
          if [ "$1" = "original_" ] && [ -f "$1"$name ];then
#check if there is the original file, if there is no need to
change it,
               printf "%s → Original file exists, continue \n"
"${IP_adress[i]}"
# otherwise we create it
          else
               printf "Trying to retrieve data from %s \n"
"${IP_adress[i]}"
               wget –q –t 2 –T 20 "${IP_adress[i]}" –O $name
#retrieve data from the ip
               wget_output[i]=$?
# keep the wget_output in case of error
               if [ "${wget_output[i]}" –ne 0 ]; then
#if wget output is not 0 then there is error
                    printf "!!! Problem in accessing IP_adress:
%s , Reason of wget failure : %d, Abort this Ip 7ddress
connection, proceed to next one\n" "${IP_adress[i]}"
"${wget_output[i]}"
               else
                    echo "Successful connection"
#if we connect, we copy all the lines except from the 1st(it has
no data) and save them to the original_$name according to the IP.
We save only the time (1st column) and the ID(3rd column) of the
sensors.
                      awk ‘BEGIN {RS = “\n”};NR!=1 {print $1,
$3}’ < $name > “$1”$name
#we delete the temporary file which was created after the wget
and keep only the original_$name
                      rm $name
                fi
          fi
     done

}
```

From this stage and on, the loop begins

```
#infinite loop
while true
do
```

2. Initialization of important parameters,

```
#initialize important parameters
     initialize


#initialize wget_output in case it will return error (page not
found/not loading/etc)
#initialize the missing error variable in case we don’t have the
appropriate number of sensors in the corresponding Ip_adress we
have an error
#initialize the array for keeping the id of each missing/non
updated sensor
initialize(){

     for i in 1 2 3 4 5 6 7 8 9 10 11 12
     do
          wget_output[i]=0
          missing_err[i]=0
          missing_sensors[i]=””
          non_updated_sensors[i]=””
     done

}
```

3. Wait for some time to read again

```
#wait for 30 minutes until we have results
        echo “Waiting for 30 minutes to check data again”
        sleep 1800
```

4. Reading of IP addresses and saving them in a temporary file

```
#after the 30 minutes take and edit new data
        edit $”check_”
```

The edit function does the same thing like the step 1. But now the name of the output files are the check_142/…/152 and not original_142.

5. Comparison between the two files original_ and check_ . Firstly, we check if for the specific arduino/IP address the connection was possible otherwise we don’t have data. Secondly, we check if the number of IDs in the original_ and check_ files are the same or if there are missing sensors we save them in the missing_err[i]. Thirdly, we check if there are exactly same lines in the files which means same ID and same time which means non updated sensor.

```
#after we have taken new correct data compare the original and
new file
        compare

#compares two files and keeps the records which have same id and
same time --> sensors has stopped updating
compare (){

        for i in 2 3 4 5 7 8 9 10 11 12
        do
                let j=$i+140
#first we check if there was an error during the retrieving of
data
                if [ "${wget_output[i]}" -ne 0 ]; then
                        printf "Can't check for non-updated or missing
sensors as there was no connection\n"
                else
#first we measure the number of sensors in check_ and then in
original_ and assign these values to missing_err[i] and len[i]
respectively
                        missing_err[i]=$(awk 'BEGIN {RS = "\n"};END
{print NR}' < "check_"$j)
                        len[i]=$(awk 'BEGIN {RS = "\n"};END {print NR}'
< "original_"$j)
#if these values are not equal and the len[i] is larger, we find
the records that are missing from the check_
                        if [ "${missing_err[i]}" -ne "${len[i]}" ]; then
```

```
                      missing_sensors[i]=$(awk
'FNR==NR{a[$2]++;next} !($2 in a) {print $2}' "check_"$j
"original_"$j )
                      printf "!!! IP adress: %s --> Missing
Sensor(s): %s\n" "${IP_adress[i]}" "${missing_sensors[i]}"
                fi
#after we check for non updated sensors which means that there
are same IDs and same records of time.
                non_updated_sensors[i]=$(awk
'FNR==NR{a[$1,$2]++;next}($1,$2) in a {print $1,$2}' "check_"$j
"original_"$j)
          fi
     done
}
```

6 Update and copy the new data.

```
#update the new data
     copy

#copy the new_IP data (=id+timestamp of sensor) to the old if and
only if we are sure the new data we took are complete and correct
(not missing sensor,page found)
copy(){

     for i in 2 3 4 5 7 8 9 10 11 12
     do
          if [ "${wget_output[i]}" -eq 0 ]; then
                let j=$i+140
                cat "check_"$j > note
#find the ID in check_ file that doesn't exist in the original_
and save it in note file. After we copy the note to original and
delete the note file.
                (awk 'FNR==NR{a[$2]++;next}!a[$2] {print $1,$2}'
"check_"$j "original_"$j) >> note
                cat note > "original_"$j
                rm "check_"$j note
          fi
     done

}
```

7 Creation of log file. During the creation of the log_file , some messages declaring if there was a problem are also shown in the terminal.
At the log_file we write the results for each one of the IP_addresses; wget_output which is if there was connectivity problem, missing sensors, non_updated sensors and the number of missing sensors if there were any.

```
#create log_file
      create_log_file

#create log file output
create_log_file(){
      (
      printf "$(date)\n"
      echo "-----------------------------------------------------
---- "
      printf "\n\n"

      for i in 2 3 4 5 7 8 9 10 11 12
      do
                  printf "IP_adress = %s ---> " "${IP_adress[i]}"
                  if [ "${wget_output[i]}" -eq 0 ] && [
"${missing_sensors[i]}" = "" ] && [ "${non_updated_sensors[i]}" =
"" ];then
                        printf "Everything was ok\n"
                  else
                        printf "There was problem (wget didn't work
or missing or non updated sensor). Check underneath for more
details about the source of the problem.\n"
                  fi
      done

      printf "\n\n"

      for i in 2 3 4 5 7 8 9 10 11 12
      do
                  printf "IP_adress = %s \n" "${IP_adress[i]}"
                  printf "Broken connection, wget output ( 0 = no
problem) = %s\n" "${wget_output[i]}"
                  let nik=${len[i]}-${missing_err[i]}
                  printf "Missing Sensors %s - %s  = %s ---> %s
\n" "${len[i]}" "${missing_err[i]}" "$nik"
"${missing_sensors[i]}"
                  printf "Sensors which haven't been updated =
%s\n" "${non_updated_sensors[i]}"
                  printf "\n"
      done
      printf "\n\n"
      ) >> log_file

#the output for the terminal
      for i in 2 3 4 5 7 8 9 10 11 12
      do
                  printf "IP_adress = %s ---> " "${IP_adress[i]}"
                  if [ "${wget_output[i]}" -eq 0 ] && [
"${missing_sensors[i]}" = "" ] && [ "${non_updated_sensors[i]}" =
"" ];then
```

```
                        printf "Everything was ok\n"
                  else
                        printf "Problem. Check log file.\n"
                  fi
      done
      printf "\n"
}
```

8	We loop to stage 2.

1.3 <u>Conclusions</u>

To sum up, the above code is for the control script of the sensors and the arduinos in the CMS cavern. In the beginning of the code 10 original files are created (original_142/…/152) in the same folder with the script in order to be used as references for the comparison with the temporary ones. If any sensor will be connected afterwards, it will be detected by the script on the next loop and it will be added automatically to the original files for the control loop. During the loop, some temporary files are created (check_142/…/152, note) in the same directory which will be deleted by the end of the loop.

So it is proposed that the script will be placed in a special folder so that the files created will be added in the same directory. If some sensors will be removed manually by the user, you have to delete all the original files and run again the script otherwise it will insist that there are some missing sensors. The format of the output of the log_file may be modified as the user wants by changing the create_log_file function.