ISSN/2499-5347

D. SURIANO

Dipartimento Sostenibilità, Circolarità e Adattamento al Cambiamento Climatico dei Sistemi Produttivi e Territoriali
Divisione Modelli, Osservazioni e Scenari per il Cambiamento Climatico e la Qualità dell'Aria
Laboratorio Modelli e Misure per la Qualità dell'Aria ed Osservazioni Climatiche
Centro Ricerche Brindisi

DSLogger: uno strumento software per l'acquisizione di dati provenienti da dispositivi e strumenti eterogenei

RT/2026/2/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE, L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

D. SURIANO

Dipartimento Sostenibilità, Circolarità e Adattamento al
Cambiamento Climatico dei Sistemi Produttivi e Territoriali
Divisione Modelli, Osservazioni e Scenari per il
Cambiamento Climatico e la Qualità dell'Aria
Laboratorio Modelli e Misure per la Qualità
dell'Aria ed Osservazioni Climatiche
Centro Ricerche Brindisi

DSLogger: uno strumento software per l'acquisizione di dati provenienti da dispositivi e strumenti eterogenei

RT/2026/2/ENEA

AGENZIA NAZIONALE PER LE NUOVE TECNOLOGIE,
L'ENERGIA E LO SVILUPPO ECONOMICO SOSTENIBILE

DSLogger: uno strumento software per l' acquisizione di dati provenienti da dispositivi e strumenti eterogenei

D. Suriano

Riassunto
Durante le più disparate attività di laboratorio, può manifestarsi l'esigenza di raccogliere le misurazioni di diversi tipi di strumenti, dispositivi, o sensori, in un unico file o database, strutturato in "records", o righe, ciascuna contenente le misure fornite dai diversi dispositivi effettuate in maniera sincrona. Ottenere ciò implica il dover collegare i dispositivi coinvolti a un computer o macchina che a intervalli regolari di tempo effettui le loro letture e le memorizzi in un file. A tal fine, occorre collegare hardware caratterizzato da diverse interfacce (USB, Ethernet, GPIO) e da differenti protocolli di comunicazione. La situazione si complica ulteriormente se si è costretti ad usare dispositivi datati o addirittura obsoleti e non supportanti il protocollo SCPI. DSlogger vuole essere un tentativo per dare una risposta a tale problematica. La sua struttura modulare consente di interfacciarsi e comunicare con uno specifico dispositivo tramite il relativo modulo o "driver", permettendo così, di poter interfacciare dispositivi eterogenei. Altre sue caratteristiche salienti sono la semplicità di utilizzo e il suo codice "open-source" che garantisce ampia flessibilità e nessun costo relativo a licenze o similari, eventualità che invece deve essere considerata nel caso si voglia utilizzare software commercialmente disponibile come, ad esempio, LabVIEW. Essendo stato sviluppato in linguaggio Python, ed essendo già disponibili strumenti di intelligenza artificiale in grado di creare codice Python, uno dei suoi possibili sviluppi futuri è la creazione automatica di driver per nuovi dispositivi utilizzando proprio tali strumenti.

Parole chiave: acquisizione dati, sensori, integrazione di dispositivi, software per acquisizione dati, misure integrate, strumenti di laboratorio.

Abstract
During a wide range of laboratory activities, the need may arise to collect measurements from different types of instruments, devices, or sensors into a single file or database, structured into "records" or rows, each containing the synchronously acquired readings provided by the various devices. Achieving this result requires connecting all the devices to a computer or system that periodically reads their data and stores them in a file. To do so, hardware with different interfaces (USB, Ethernet, GPIO) and different communication protocols must be connected. The situation becomes even more complex when one is forced to use outdated or even obsolete equipment that does not support the SCPI protocol. DSlogger aims to address this challenge. Its modular structure allows communication with a specific device through its corresponding module or "driver," enabling the integration of heterogeneous devices. Other key features include ease of use and its open-source code, which ensures high flexibility and eliminates any costs related to licenses or similar constraints. These issues have to be considered if we mean to use commercially available software, such as, for example, LabVIEW. By being implemented through Python language and by being already available artificial intelligence tools capable of creating Python code, one of its possible future developments could be the automatic building of drivers for new devices by using such tools.

Keywords: *data acquisition, sensors, device integration, data acquisition software, integrated instruments, laboratory instruments.*

INDICE

1. Introduzione

Le tipiche attività di laboratorio possono prevedere l'impiego di strumentazione eterogenea e, in alcuni casi, l'utilizzo contemporaneo di sensori e di strumenti di misura professionali. Una simile situazione si verifica tipicamente quando si vuole calibrare, o semplicemente valutare, sensori per la qualità dell'aria. Tale attività dovrà necessariamente prevedere la lettura sincrona dei sensori e delle misure fornite dagli strumenti di riferimento che andrà a costituire un "record" da memorizzare in un file, o database, per successive elaborazioni o analisi.
Il primo problema da affrontare è quindi rappresentato dal dover collegare a un computer dell' hardware di tipo eterogeneo, caratterizzato da interfacce di diverso tipo e da specifici protocolli di comunicazione. Nel caso di strumenti professionali abbastanza recenti, in genere, tale problematica è risolta con l'uso di interfacce USB, Ethernet o GPIB sulle quali viene utilizzato lo standard di comunicazione SCPI [1,2]. Tale protocollo è basato sull' invio di comandi costituiti da stringhe di caratteri ASCII e sulla ricezione di stringhe similari contenti l'output, come ad esempio, la misurazione effettuata dallo strumento.
La gestione del sistema di acquisizione dati che poggia su tale architettura è affidata a software commercialmente disponibili come LAbview, o Matlab. Nel caso di LAbview, la gestione corretta del flusso di stringhe contenenti comandi SCPI è affidato a librerie VISA integrate nel software [3]. Un ambiente grafico facilita l'uso di tali librerie con le quali è possibile costruire il "driver", o modulo, o strumento virtuale, che si interfaccia tra l'utente finale e il dispositivo da monitorare.
Nel caso di Matlab, l'invio e la ricezione di comandi SCPI è realizzato scrivendo un codice, o script, che sfrutta delle funzioni messe a disposizione dall' "Instrument Control Toolbox" integrato nel software [4]. Per controllare quindi uno o più strumenti, sarà necessario scrivere uno script opportuno che dovrà essere eseguito in ambiente Matlab. La stesura di tale script necessita però di competenze specifiche inerenti la sintassi del linguaggio Matlab.
Vi sono casi in cui il dispositivo non supporta comandi SCPI, magari perché risulta essere un po' datato, o semplicemente perché supporta protocolli proprietari. E' questo sicuramente il caso di alcuni sensori "low-cost", o di alcune apparecchiature come, ad esempio, il monitor per gli ossidi di azoto "405nm" della 2Btech. Per controllare questi dispositivi occorre conoscere i comandi che costituiscono il protocollo proprietario e gestire il loro flusso attraverso l'interfaccia hardware prevista che in genere è la porta seriale RS232, USB o la porta Ethernet. In tali casi, bisogna scrivere i comandi direttamente sulla porta di comunicazione relativa, e occorre conoscere il protocollo di comunicazione specifico del dispositivo. Per effettuare tale operazione, LabVIEW mette a disposizione un ambiente grafico che, in un certo senso, facilita il compito. Per quanto riguarda Matlab, ci si può affidare alle funzioni relative contenute nella apposita libreria sulle quali sarà possibile scrivere lo script che gestirà il monitoraggio dello strumento.
Nel caso si voglia utilizzare questi software, bisogna considerare che essi non sono gratuiti. In entrambi i casi, ci sono versioni con licenza gratuita con funzionalità limitate o con caratteristiche che ne limitano il loro utilizzo.

2. Caratteristiche e descrizione del software

Il software DSlogger ha un codice sorgente scritto interamente in Python [5], ed è quindi di tipo "open-source". Tale caratteristica ne garantisce la gratuità e l'accesso completo ai file sorgenti offrendo ad un utenza con le dovute competenze la possibilità di modificare o aggiungere le funzionalità di base. Oltre tale possibilità, è previsto che l'utente finale possa scrivere da se nuovi moduli o drivers specifici per nuovi strumenti o dispositivi . La loro scrittura dovrà però seguire alcuni vincoli che verranno esaminati dettagliatamente in seguito. Il linguaggio Python con cui è scritto il

codice sorgente del software offre anche il vantaggio di permettere l'utilizzo di strumenti di intelligenza artificiale (AI) per scrivere i nuovi drivers. Infatti, diverse piattaforme AI gratuite e disponibili sul web, come ad esempio Chatgpt [6,7] o Google Gemini [8,9] sono in grado di generare codice Python fornendo loro istruzioni scritte o addirittura indicandole a voce.
Sono stati sviluppati due pacchetti di installazione per DSlogger: il primo consente la sua installazione su macchine a 64 bit su cui gira il sistema operativo Windows 11, il secondo invece consente l'installazione su macchine più vecchie a 32 bit su cui sono presenti sistemi operativi più datati, come Windows XP o Windows 7. Questo secondo pacchetto consente l'utilizzo di PC datati o obsoleti, ma perfettamente funzionanti, che altrimenti, costituirebbero un prodotto di rifiuto anche se privi di guasti o difetti.
L'interfaccia utente è di tipo testuale a linea di comando come raffigurato in figura 1. Essa è composta da due finestre di cui una è adibita esclusivamente a visualizzare le misurazioni che vengono fatte in tempo reale (Data Viewer o DV), mentre l'altra (Command Window CW) consente di immettere i comandi e di visualizzarne le risposte. L'output del software consiste quindi nei dati mostrati nella DW e in un file di testo tipo CSV (Comma Separated Values) in cui vengono memorizzate le misurazioni lette dai diversi dispositivi in quel momento connessi, insieme alla data e ora della loro rilevazione. La struttura e le caratteristiche di questo file verranno riprese dettagliatamente in seguito.
In questa prima versione del software sono stati sviluppati i driver relativi a 11 diversi strumenti e dispositivi, mentre il set di comandi è composto da 8 diversi comandi.

Figura 1: l'interfaccia utente del software DSlogger.. A sinistra è mostrata la finestra DV, mentre a destra la CW.

3. La struttura del software

Il codice del software è organizzato in due files principali, il "DSlogger.py", e il "data_viewer.py" . A loro complemento vi sono i files di driver contenuti nella directory "devices" che dovrà necessariamente essere nella stessa directory in cui sono posizionati i due files principali per un corretto funzionamento del software. Nella directory principale troviamo la directory "data" dove DSlogger crea i files CSV nei quali vengono memorizzate le misurazioni. Il file "installed.ini", che è un file di testo, tiene traccia dei dispositivi correntemente installati nel sistema. Il resto dei files o directory presenti nella directory principale sono invece risorse necessarie per il processo di installazione del software nel sistema operativo del PC. Il funzionamento dell'applicazione può essere sintetizzato nel diagramma di flusso mostrato in figura 2.

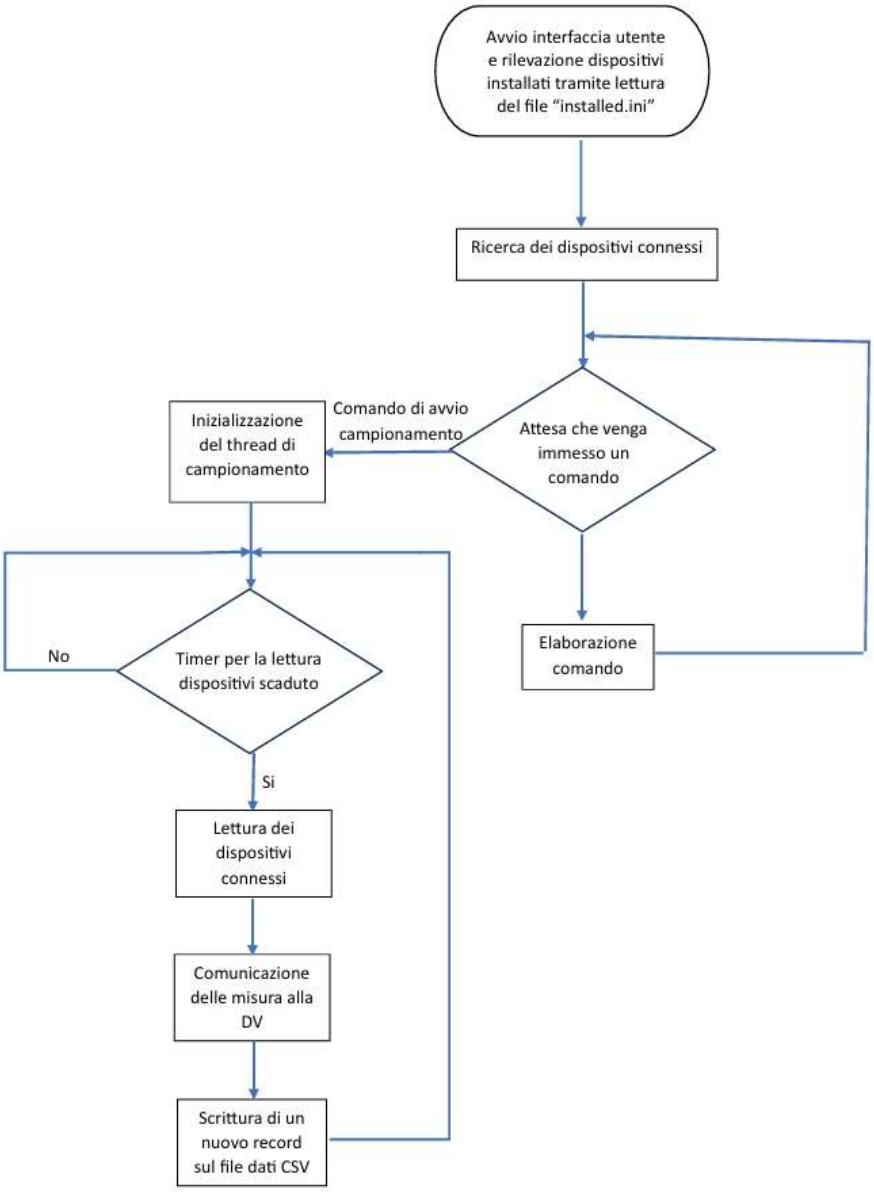

Figura 2: diagramma di flusso sintetico del software DSlogger.

DSlogger è un applicazione che ha due processi principali eseguiti in contemporanea. Il primo viene codificato nel file "DSlogger.py", costituisce la spina dorsale dell' applicazione, e ha un interfaccia utente implementata dalla finestra CW. Il secondo processo è codificato nel file "data_viewer.py" ed ha come unico scopo la visualizzazione delle misure effettuate in tempo reale, pertanto, gestisce la finestra DV. La comunicazione tra i due è di tipo unidirezionale e avviene sfruttando socket UDP secondo uno schema "master-slave", quindi, i dati costituiti dalle misure dei dispositivi connessi, letti dal processo principale (master), vengono comunicati al processo che potremmo definire "data viewer" (slave) , che avrà l'unico compito di visualizzarle nell'apposita finestra. All'avvio, viene avviato il processo principale, che a sua volta, avvierà il "data viewer" tramite la funzione "viewer_opening()" (vedi codice del file "DSlogger2.0.py" in allegato a questo documento). La seconda operazione importante in avvio è costituita dalla lettura del file "installed.ini" che tiene traccia dei driver attualmente installati nell'applicazione. La lettura di questo file permetterà che tali driver vengano caricati nella memoria allocata per il processo principale e, in ultima analisi, permetterà l'utilizzo dei dispositivi ad essi relativi. Questa operazione è effettuata dalla funzione "load_devices()".
Il passo successivo eseguito dall'applicazione è la verifica di quali dispositivi siano effettivamente connessi tra quelli installati. Questa operazione viene effettuata interrogando tutte le porte a disposizione del PC (USB o Ethernet). La funzione dedicata a questo compito è la "device_scanning(conn_dev,dev)", la quale restituisce un' array di oggetti rappresentanti i dispositivi individuati.
Esaurite le operazioni di avvio, l'applicazione entra in un "loop" bloccante col quale si mette in attesa di un comando digitato nella finestra CW. L'elaborazione dei comandi, dal punto di vista del flusso dati, non si differenzia molto tra un comando e l'altro, tranne nel caso in cui si voglia iniziare una sessione di campionamento dei dispositivi connessi. In questo caso, il processo "madre" ha bisogno di generare un "thread" che deve essere eseguito in contemporanea col "thread" principale costituito dal "loop" bloccante menzionato pocanzi. Infatti, bisogna assicurare che in contemporanea il software sia in grado di leggere gli output dei dispositivi, visualizzarli nella DV, e memorizzarli nel file CSV, senza interrompere le funzionalità di ricezione e esecuzione dei comandi inseriti dall' utente. Il "thread" generato quando viene avviata una nuova sessione di campionamento è codificato nella funzione "capture(sk1,conn_dvc,appdir)".

4. Utilizzo del software

Nella versione attuale del software sono stati scritti i driver per i dispositivi elencati nella tabella 1.

Dispositivo	Descrizione	Interfaccia	Nome file driver
SPS30 [10]	*Sensore miniaturizzato per le polveri sottili*	*UART*	*sps30.py*
PMS5003 [11]	*Sensore miniaturizzato per le polveri sottili*	*UART*	*pms5003.py*
405nm [12]	*Monitor per gli ossidi di azoto*	*RS232*	*nox405.py*
MHZ19 [13]	*Sensore miniaturizzato per CO_2*	*UART*	*mhz19.py*
IRCA1 [14]	*Sensore miniaturizzato per CO_2*	*USB*	*irca1.py*
GO3 [15]	*Monitor per ozono e anidride carbonica*	*RS232*	*go3.py*
V72M [16]	*Analizzatore chimico per VOC*	*Ethernet*	*v72m.py*
O342 [17]	*Analizzatore chimico per ozono*	*Ethernet*	*o342.py*
CO12M [18]	*Analizzatore chimico per monossido di carbonio*	*Ethernet*	*co12m.py*
AF22 [19]	*Analizzatore chimico per biossido di zolfo*	*Ethernet*	*af22.py*
AC32 [20]	*Analizzatore chimico per biossido di azoto*	*Ethernet*	*ac32.py*

Tabella 1: sensorio strumenti i cui driver sono già disponibili nella corrente versione del software DSlogger.

Una precisazione va fatta per quanto riguarda gli strumenti utilizzabili con la presente versione del software tramite interfaccia Ethernet. Allo stato attuale, il modo corretto di utilizzare un qualsiasi strumento o dispositivo interfacciabile tramite Ethernet è quello che prevede la creazione di una rete locale impostando un IP statico al PC su cui viene eseguito il software. Tale PC sarà collegato a uno o più strumenti tramite un Ethernet "Hub" che è un dispositivo facilmente reperibile in commercio (vedi ad esempio [21]). Tutti gli strumenti andranno connessi all' "Hub" e avranno a loro volta un IP statico appartenente alla stessa rete del PC. Per meglio chiarire quanto detto fin qui, in tabella 2 è mostrato un esempio reale di configurazione di rete con gli IP assegnati a ciascuna macchina che la compone.

Dispositivo	**IP statico assegnato**
PC	*192.168.20.10*
V72M	*192.168.20.40*
0342	*192.168.20.44*
CO12M	*192.168.20.41*
AF22	*192.168.20.42*
AC32	*192.168.20.43*

Tabella 2: una configurazione suggerita per utilizzare gli strumenti interfacciabili tramite Ethernet.

Come si evince dalla tabella 2, i dispositivi appartengono tutti alla stessa rete locale "192.168.20.xxx". Ovviamente, è possibile specificare indirizzi IP diversi modificando nei rispettivi "driver" il valore della variabile "ETH_ADDRESSES" e impostando l'hardware di ciascun strumento con il medesimo IP. Il dispositivo Ethernet "Hub" non ha bisogno di driver per sistema operativo da installare sul PC per poter funzionare correttamente.

Con riferimento alla tabella 1, gli strumenti 405nm e GO3 prodotti dalla 2Btech sono provvisti di interfaccia RS232. Essi sono utilizzabili dai PC che non posseggono tale interfaccia usando un adattatore RS232/USB facilmente reperibile in commercio [22]. I sensori miniaturizzati SPS30, PMS5003 e MHZ19 affidano i loro dati di output all' interfaccia UART con livello logico alto pari a 3,3V. Anche in questo caso, per poterli connettere ad un PC, sarà sufficiente usare un adattatore UART/USB tra i diversi modelli disponibili sul mercato, ad esempio, il USB-RS232-PCB prodotto dalla FTDI [23]. L'uso di adattatori di interfaccia RS232/USB o UART/USB potrà prevedere l'installazione di un driver relativo che è però distribuito gratuitamente dalle rispettive aziende produttrici.

In ultima analisi, i dispositivi che non hanno un interfaccia USB, ma che comunque hanno un interfaccia di tipo seriale, come la vecchia RS232 o la UART, possono sempre essere usati con adattatori facilmente reperibili in commercio, che li rendono comunque utilizzabili con DSlogger. I dispositivi che hanno interfaccia USB, RS232 o UART vengono quindi considerati dal sistema DSlogger come un unico tipo di dispositivo che si interfaccia su porta di tipo seriale.

Il primo passo per poter utilizzare DSlogger è costituito dalla sua installazione sul PC. Nel caso si abbiano macchine a 32 bit su cui gira Windows XP o Windows 7 occorre far riferimento al pacchetto "DSlogger1.0" [24]; esso è contenuto nell'omonima cartella che dovrà essere copiata in una qualsiasi posizione (ovvero cartella) all'interno del proprio PC, e ciò costituirà il primo passo per poter installare il software. Al suo interno troviamo il file "install.bat" che deve essere eseguito con la usuale manovra del doppio "click" sul pulsante sinistro del mouse. Tale file contiene uno script di comandi MS-DOS che esegue dapprima il file "python-2.7.8.msi", col quale si installeranno le principali librerie "python 2.7.8", poi il file "pyserial-2.7.win32.exe", che invece installerà le librerie "python" per poter gestire le porte

USB. Tutta la procedura guiderà passo per passo l'utente a completare l'installazione del software in modo semplice ed intuitivo come evidenziato dalle figure 3,4,5,6,7 e 8.

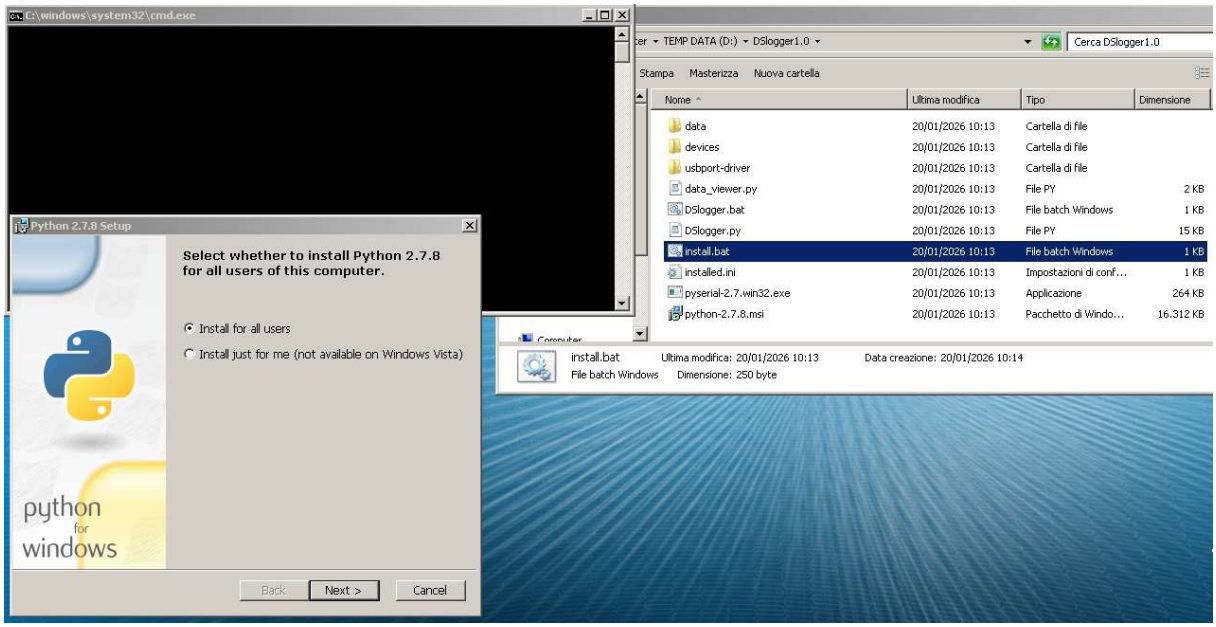

Figura 3: per avviare l'installazione del software bisogna eseguire il file "install.bat" visualizzato in figura. Esso avvierà l'esecuzione del file "python-2.7.8.msi". Il risultato finale di questo primo passo del processo di installazione è rappresentato in questa figura. Per procedere l'utente dovrà premere sul pulsante "Next".

Figura 4: Il passo successivo a quello evidenziato in figura 3 è rappresentato in figura. In questa fase, l'utente può scegliere in quale posizione verranno installate le librerie Python. Per procedere l'utente dovrà premere sul pulsante "Next".

Figura 5: Il passo successivo a quello evidenziato in figura 4 è rappresentato in figura. In questa fase, l'utente deve selezionare l'opzione "Add python.exe to Path" necessaria per il corretto funzionamento del software. Per procedere l'utente dovrà cliccare sulla casella omonima visualizzata in figura.

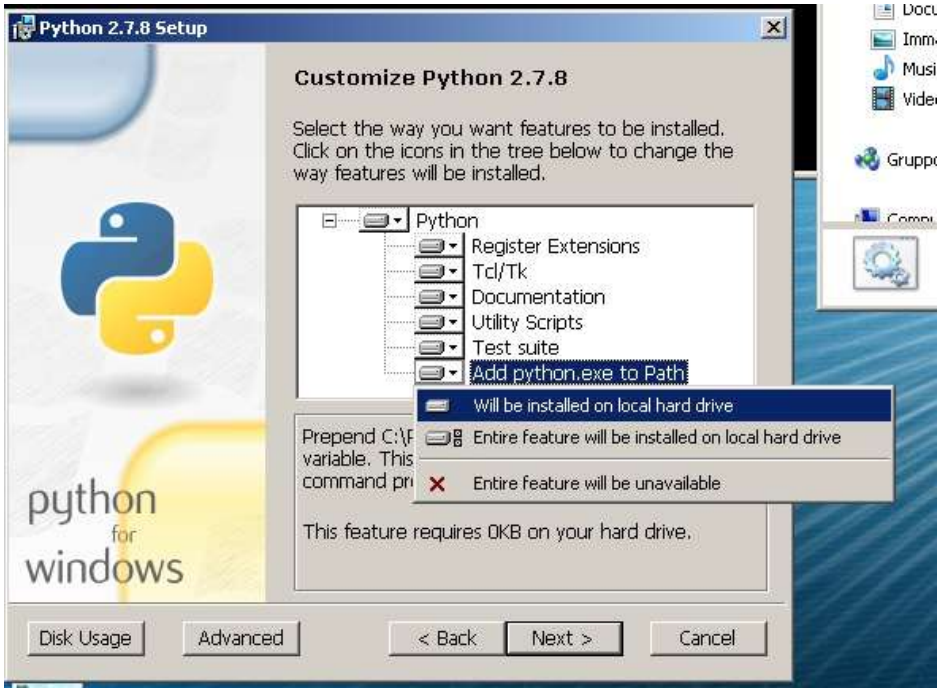

Figura 6: in questa figura è evidenziata la necessaria l'opzione "Add python.exe to Path". Una volta selezionata questa opzione, l'utente dovrà cliccare sul pulsante "Next" per procedere.

Figura 7: in questa figura è mostrata la maschera che viene visualizzata quando l'installazione del linguaggio Python è andata a buon fine. Premendo il pulsante "Finish", il file "install.bat" richiamerà la successiva procedura di installazione che avvierà il file necessario per installare le librerie pyserial-2.7 che gestiscono le porte seriali e USB. In questa procedura è consigliabile che l'utente si limiti solo a cliccare sul pulsante "Avanti" senza modificare le impostazioni suggerite dalle maschere.

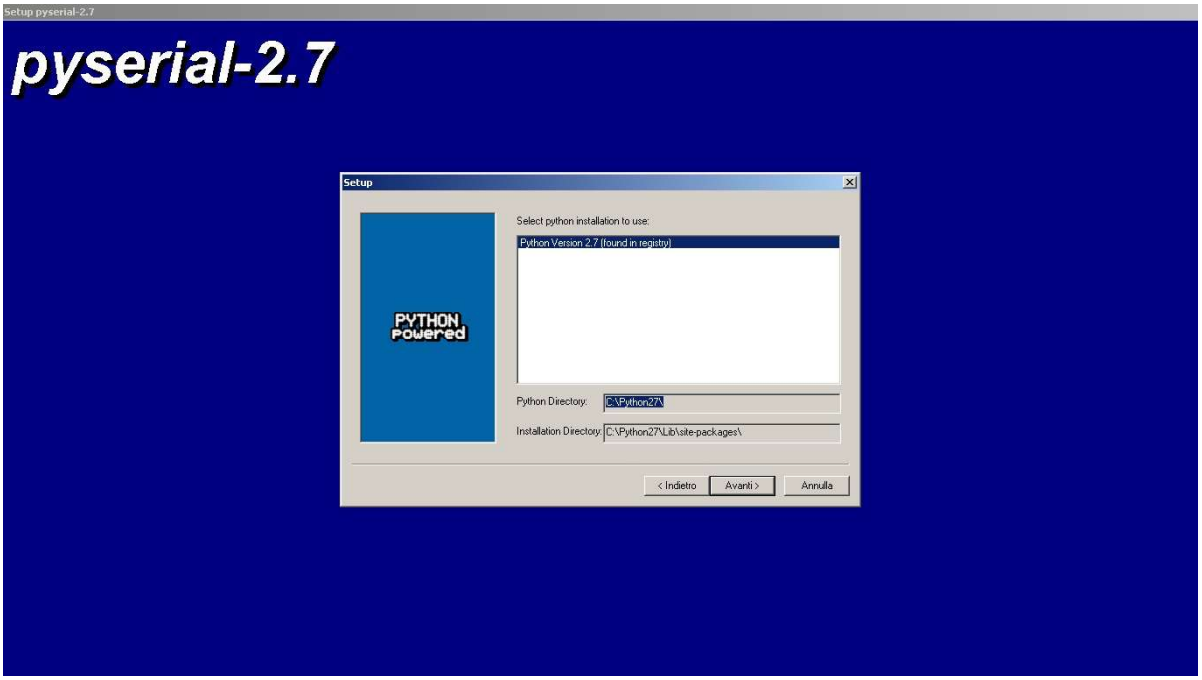

Figura 8: in questa figura è mostrata una delle maschere relative all'installazione delle librerie Pyserial-2.7 necessarie per la gestione delle porte seriali e USB. In questa fase è consigliabile che l'utente clicchi solamente sul pulsante "Avanti" fino a completare la procedura stessa.

Figura 8: terminata l'installazione delle librerie pyserial-2.7, lo script "install.bat" mostra la finestra dove vengono ricapitolate le due procedure di installazione. L'utente a questo punto dovrà digitare un tasto qualsiasi per poter permettere allo script il riavvio automatico del PC necessario per completare l'installazione del software.

Se si ha a disposizione un PC con sistema operativo a 64 bit (ad esempio, Windows 10 o 11), è necessario utilizzare il pacchetto "DSlogger 2.0" [25] che non prevede alcuna particolare procedura di installazione. Sarà quindi sufficiente copiarlo in una qualsivoglia posizione del proprio PC ed eseguire il file "DSlogger2.0.exe" che avvierà il software, viceversa, per l'avvio nella versione "DSlogger1.0" da utilizzare su macchine a 32 bit, bisognerà eseguire il file "DSlogger.bat".

All'avvio, coerentemente con il diagramma di flusso presentato in figura 2, viene letto il file "installed.ini", dove è scritto quali driver tra quelli disponibili nella cartella "devices" verrà caricato nella memoria riservata all'applicazione. Caricati i drivers, il software provvederà a scansionare le porte di comunicazione per verificare quali e quanti dispositivi siano effettivamente collegati al PC. Terminata questa fase, viene mostrato l'elenco dei comandi disponibili per l'utilizzo del software (vedi figura 9). Un'altra operazione effettuata all'avvio è il lancio delle due finestre che costituiscono l'interfaccia utente: la CW e la DV. Terminate queste operazioni, la CW mostrerà il prompt comandi ">>>".

A questo punto, l'operazione tipica che un utente può effettuare è l'esecuzione di una sessione di monitoraggio dei dispositivi collegati al PC con una frequenza di campionamento specificata in secondi (vedi tabella 3). Una volta lanciata la sessione di monitoraggio, il software creerà un file CSV nella cartella "data" attribuendogli un nome ricavato dalla data e ora corrente. In tale file verranno memorizzati i "records" riportanti le misurazioni effettuate con data e ora del rilevamento (vedi figura 10). Durante una sessione di monitoraggio sarà possibile lanciare dalla CW solo i comandi di fine monitoraggio ("b"), o di visualizzazione del menù dei comandi ("h"), quindi, per effettuare qualsiasi altra operazione, sarà necessario fermare la sessione di monitoraggio ("b"). Durante questa fase, è consigliabile non cancellare, spostare o modificare Il file CSV nel quale vengono memorizzate le misurazioni; viceversa, sarà possibile leggerlo o copiarlo.

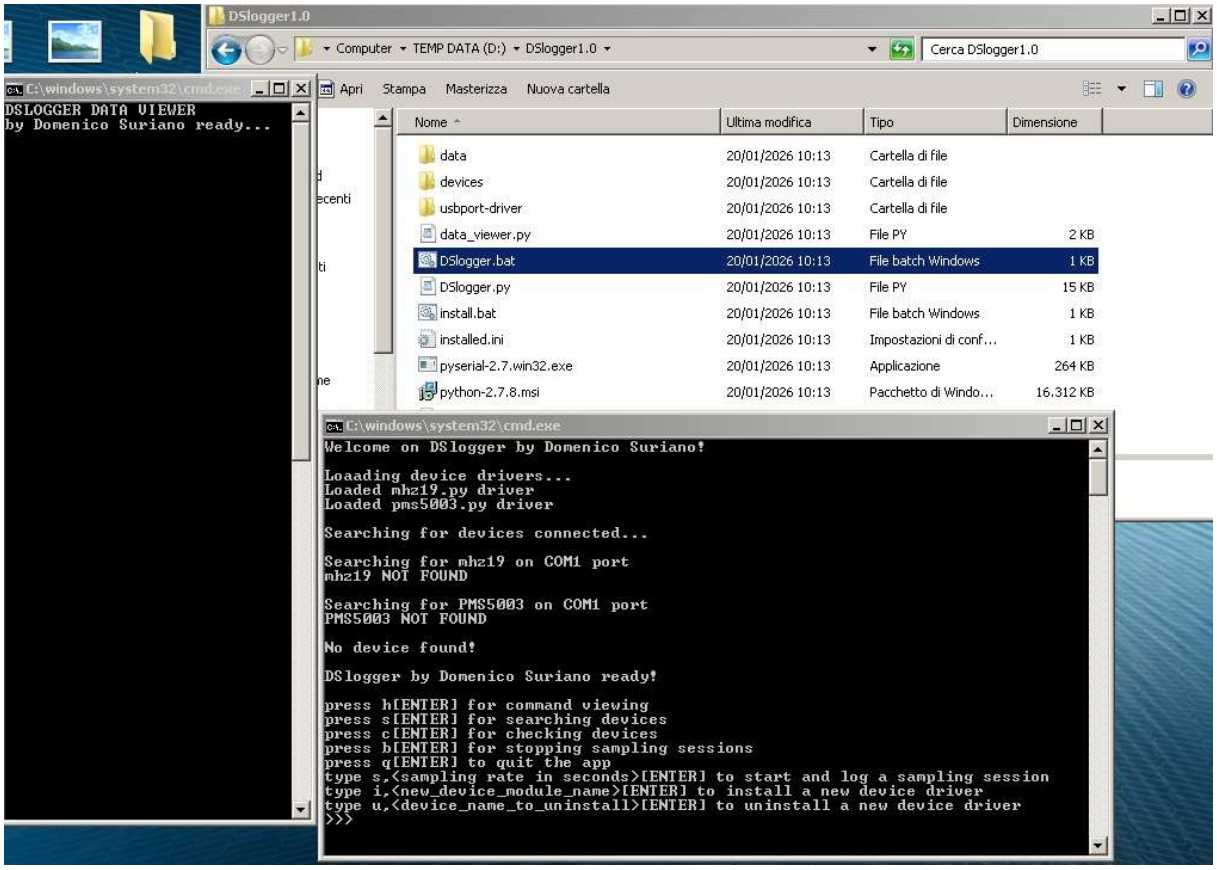

Figura 9: l'aspetto delle finestra CW e DV ad avvio terminato. Nella CW sono visibili i resoconti delle operazioni di avvio effettuate, il menù dei comandi, e il prompt ">>>".

Come visibile nella tabella 3, esistono due tipi di comandi: quelli con parametro, e quelli senza. Nel caso sia necessario specificare un parametro, il formato del comando sarà "comando,parametro" (ad esempio: s,5) senza l'inserimento di spazio nella stringa del comando. Durante una sessione di monitoraggio, le letture dei singoli dispositivi vengono visualizzate in tempo reale nella DV, che provvederà a visualizzare anche data e ora della relativa lettura.

Un'altra importante operazione è rappresentata dalla installazione o disinstallazione dei drivers dei dispositivi all'interno del software. E' opportuno specificare che i drivers a disposizione dell'utente andranno tutti posti nella cartella "devices", ma non tutti verranno effettivamente caricati al momento dell'esecuzione dell'applicazione. Solo quelli memorizzati nel file "Installed.ini" verranno caricati all'avvio e quindi effettivamente utilizzati. E' fortemente sconsigliato modificare tale file con procedure diverse da quelle che verranno attivate con i relativi comandi (ossia "i,nome del driver del dispositivo" e "u, nome del driver del dispositivo").

L'output del software è in parte rappresentato dal file CSV che viene generato ogni qualvolta si procede ad avviare una nuova sessione di monitoraggio. Esso è strutturato in campi separati da punto e virgola (";"), ed ha come prima riga i nomi delle colonne che contengono i dati dei record scritti dalla seconda riga in poi. La figura 10 chiarisce ogni dettaglio relativo a questo file.

Comando	Parametro	Descrizione
h	*nessuno*	*Visualizza nella CW il menù dei comandi*
s	*secondi*	*Il comando "s" seguito dalla virgola e da un numero intero (ad esempio: s,10) avvia una sessione di monitoraggio con frequenza di campionamento pari a 10 secondi. Ossia, tutti i dispositivi rilevati verranno letti ogni 10 secondi e i relativi valori verranno memorizzati nell'apposito file CSV.*
b	*nessuno*	*Termina e conclude una sessione di monitoraggio.*
s	*nessuno*	*Il comando "s" seguito da nessun parametro effettuerà una nuova ricerca dei dispositivi connessi col PC. Esso non è disponibile se è in corso una sessione di monitoraggio.*
c	*nessuno*	*Questo comando esegue una singola lettura dei dispositivi connessi col PC. Esso non è disponibile se è in corso una sessione di monitoraggio.*
i	*nome del driver*	*Installa il driver il cui nome è specificato nel parametro (ad esempio: "i,pms5003.py"). Esso modifica correttamente il file "installed.ini". Per rendere efficace tale comando occorrerà riavviare l'applicazione.*
u	*nome del driver*	*Disinstalla il driver il cui nome è specificato nel parametro (ad esempio: "u,pms5003.py"). Esso modifica correttamente il file "installed.ini". Per rendere efficace tale comando occorrerà riavviare l'applicazione.*
q	*nessuno*	*Chiude ed esce correttamente dall'applicazione. E' sconsigliato chiuderla semplicemente chiudendo le finestre CW e DV.*

Tabella 3: elenco dei comandi disponibili su DSlogger e loro descrizione.

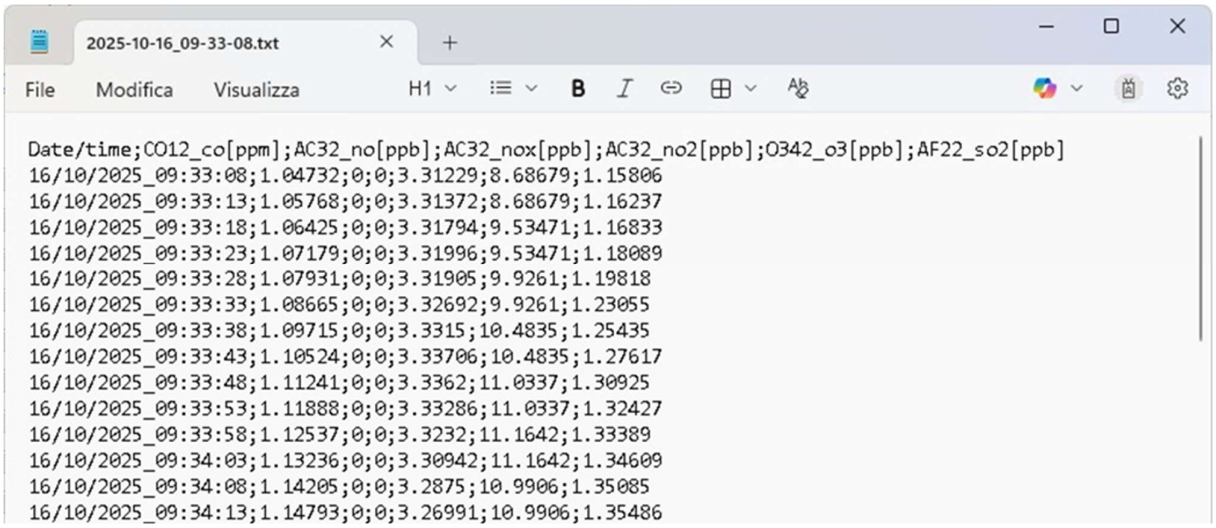

Figura 10: il file dati tipo CSV nel quale vengono memorizzate le letture dei dispostivi. Nella prima riga troviamo i nomi delle misure effettuate che vengono desunti dal driver del dispositiv1. Il software attribuisce automaticamente il nome del file leggendo l'orologio del computer nell'ora e nella data in cui viene iniziata la sessione di monitoraggio.

5. Come scrivere nuovi drivers

Un file driver è uno script python che contiene il codice per gestire uno specifico dispositivo da utilizzare con DSlogger. Esso non fa parte del corpo principale del software poiché l'utilizzo di ciascun dispositivo è relativo alle esigenze di uno specifico utente. Inoltre, non risultava pratico incorporare

nel software principale in maniera persistente ogni dispositivo utilizzabile col software, poiché, i tempi di scansione e di ricerca di tutti dispositivi potenzialmente collegati e collegabili col PC (operazione effettuata all'avvio o con il comando "s") sarebbero stati eccessivamente lunghi. Si è preferito, quindi, lasciare all'utente finale quali driver caricare nello spazio di memoria dedicato all'esecuzione di DSlogger tra quelli disponibili. Tale scelta verrà effettuata dall'utente in base ai dispositivi che effettivamente andrà ad utilizzare col comando "i,nome_del_driver". La possibilità di installare o disinstallare moduli nel software principale conferisce la necessaria flessibilità al sistema, fondamentale per poter permettere ad un utente con un minimo di competenze in scrittura di script python di poter creare il driver specifico per ogni proprio dispositivo o strumento. Per di più, con l'avvento dei nuovi strumenti software di intelligenza artificiale capaci di creare codici in python seguendo semplici indicazioni da parte dell'utente, addirittura, sarebbe possibile creare driver anche senza conoscere la sintassi del linguaggio python. Ad ogni modo, nel comporre nuovi driver, si dovranno rispettare alcune regole o vincoli affinchè il software DSlogger possa li utilizzare efficacemente e senza errori. Tali vincoli sono riassumibili nei seguenti punti:

1) Per i driver relativi a dispositivi che si interfacciano tramite porta seriale (USB, UART o vecchia RS232) è obbligatorio importare la libreria python in grado di gestire tali risorse inserendo nel codice la riga *"import serial"*. Parimenti, i driver dei dispositivi che si interfacciano tramite ethernet dovranno importare la libreria che gestisce tale tipo di comunicazione tramite l'istruzione "*import socket*". Queste operazioni possono essere implementate nelle prime righe del codice (vedi codici allegati) .
2) E' obbligatorio inserire tre costanti stringa di tipo globale e denominarle: *CONNECTION_TYPE, DEVICE_IDENTITY, DEVICE_SENSORS*. La costante *CONNECTION_TYPE* dovrà essere impostata al valore "usb" o "serial" (l'una o l'altra indifferentemente) per i dispositivi che si interfacciano tramite porta seriale; sarà invece impostata con la stringa "eth" per i dispositivi collegati tramite ethernet. La costante *DEVICE_IDENTITY* assegna il nome dato al dispositivo tramite il quale verrà identificato all'interno del sistema DSlogger. Esso è a discrezione dell'utente. La costante globale *DEVICE_SENSORS* è invece usata per immagazzinare l'informazione riguardante quali e quante misure il dispositivo fornisce al sistema. Esse saranno obbligatoriamente separate dal carattere ";". Ad esempio, l'analizzatore chimico per gli ossidi di azoto fornisce misure di NO in "ppb", di NO_2 in "ppb" e del globale NO_x sempre in "ppb"; quindi, l'utente imposterà il valore di *DEVICE_SENSORS* alla stringa "NO[ppb];NO2[ppb];Nox[ppb]". Un' altra costante globale obbligatoria è invece *DEVICE_BAUD_RATE* per i dispositivi che si interfacciano tramite porta seriale. Essa andrà impostata ad una stringa che riflette il valore di "baud rate" con cui la comunicazione seriale verrà effettuata (ad esempio: "19200"). Nel caso in cui bisogna gestire comunicazioni tramite "ethernet" è obbligatorio inserire la costante globale *ETH_ADDRESSES* che dovrà essere un' array di stringhe contenenti gli indirizzi IP ai quali il dispositivo è disponibile (ad esempio: ["192.168.20.44"], oppure: ["192.168.20.44","192.168.20.45"]) .
3) E' obbligatorio creare una classe denominata come il nome del file driver che la contiene, ma con la lettera iniziale in maiuscolo. Ad esempio, il sensore IRC-A1 ha un driver il cui nome file è "irca1.py"; in tal caso, all'interno del file bisognerà creare la classe "Irca1" tramite la riga *"class Irca1"*.
4) La classe così creata dovrà contenere i seguenti metodi pubblici riassunti nelle tabelle sottostanti.

Metodo pubblico	Valore di ritorno	Commenti
getConnectionParams(self)	*[self.portname,self.baud_rate]*	Tramite questo metodo il sistema reperisce informazioni sui parametri di connessione. Il valore di ritorno è un array di due elementi stringa. Il primo indica il nome della porta seriale, il secondo invece il valore di baud rate.
getConnectionType(self)	*self.connection_type*	Tramite questo metodo il sistema reperisce informazioni sul tipo di connessione ("*usb*" o "*serial*"). Il valore di ritorno deve essere quello impostato nella costante *CONNECTION_TYPE*
getIdentity(self)	*self.identity*	Tramite questo metodo il sistema ricava il nome del dispositivo. Il valore di ritorno deve essere quello impostato per la costante *DEVICE_IDENTITY*
getSensors(self)	*self.sensors*	Tramite questo metodo il sistema ricava quali e quante misure fornisce il dispositivo. Il valore di ritorno deve essere quello impostato per la costante *DEVICE_SENSORS*
terminate(self)	*nessuno*	Tramite questo metodo il sistema "chiude" l'oggetto creato a partire dalla classe che rappresenta il dispositivo nel software.
connect(self,serport)	*Valore intero 0 o 1*	Tramite questo metodo il sistema rileva se il dispositivo è connesso. Se esso è effettivamente connesso, il valore di ritorno sarà pari a 1, altrimenti 0. Il valore "serport" è una stringa che identifica la porta serial o USB sulla quale si sta effettuando il controllo di presenza del dispositivo.
sample(self)	*Stringa contenente i valori delle letture del dispositivo intervallate dal carattere ";"*	Tramite questo metodo il sistema comanda la lettura delle misurazioni correnti del dispositivo. Il valore di ritorno può essere ad esempio la stringa: "0.121;1.233;3.256". Si dovrà prevedere una stringa errore che verrà restituita in caso di malfunzionamento tecnico. Tale stringa pouò essere costituita da valori fuori range dello strumento, ad esempio: "-1000;-1000;-1000"

Tabella 4: elenco dei metodi da inserire in una classe relativa a un dispositivo che si connette tramite porta di tipo seriale.

Metodo pubblico	Valore di ritorno	Commenti
getConnectionParams(self)	*["IP_address1";"IP_address2"]*	Tramite questo metodo il sistema reperisce informazioni sui parametri di connessione. Il valore di ritorno è un array di elementi stringa contenente gli indirizzi IP ai quali è possibile comunicare con il dispositivo. Tale array deve essere quello impostato per la costante *ETH_ADDRESSES*
getConnectionType(self)	*self.connection_type*	Tramite questo metodo il sistema reperisce informazioni sul tipo di connessione ("eth"). Il valore di ritorno deve essere quello impostato nella costante *CONNECTION_TYPE*
getIdentity(self)	*self.identity*	Stesse considerazioni relative alla tabella 4
getSensors(self)	*self.sensors*	Stesse considerazioni relative alla tabella 4
terminate(self)	*nessuno*	Stesse considerazioni relative alla tabella 4
connect(self,address)	*Valore intero 0 o 1*	Tramite questo metodo il sistema rileva se il dispositivo è connesso. Se esso è effettivamente connesso, il valore di ritorno sarà pari a 1, altrimenti 0. La variabile "address" è una stringa contenente l' indirizzo IP sul quale si sta controllando se il dispositivo sia connesso.
sample(self)	*Stringa contenente i valori delle letture del dispositivo intervallate dal carattere ";"*	Stesse considerazioni relative alla tabella 4

Tabella 5: elenco dei metodi da inserire in una classe relativa a un dispositivo che si connette tramite ETHERNET.

A maggiore chiarezza di quanto finora esposto fin qui, andremo a commentare ed esaminare i codici dei driver relativi a due dispositivi, il sensore PMS5003 e l'analizzatore chimico AC32. Esaminiamo dapprima il caso del sensore PMS5003. Esso ha una interfaccia UART con logica 0-3.3V, pertanto, così come acquistato dal produttore, non potrebbe essere utilizzato con un PC, e quindi, con DSlogger. Tuttavia, collegandolo, per esempio, all'adattatore UART/USB USB-RS232-PCB della FTDI, può essere utilizzato col nostro sistema. Allo scopo, sarà necessario importare le librerie Pyserial in grado di gestire le porte USB o RS232 dei PC. Infatti, alla riga 1 dell'allegato riportante il codice del driver per PMS5003 troviamo l'istruzione "*import serial*" che effettua tale operazione. Va chiarito che le librerie PYserial gestiscono le porte seriali (USB o RS232) indifferentemente vedendole come porte virtuali "COM" su sistemi operativi Windows. Nelle righe 4-7 troviamo la dichiarazione delle costanti obbligatorie esplicitate nel punto 2). Notiamo che la costante *DEVICE_SENSORS* è impostata alla stringa "pm1[ug/m3];pm2.5[ug/m3];pm10[ug/m3]". Infatti, il sensore PMS5003 fornisce come output diverse misure, tra cui una stima della concentrazione di PM10, PM2.5 e PM1 espresse in ug/m^3. Essendo interessati a rilevare unicamente queste letture, impostiamo la costante di conseguenza, separando le misurazioni con il carattere ";". Avremmo potuto anche impostarla a "PM1;PM2.5;PM10" in

quanto la denominazione della singola misura è a discrezione di chi scrive il codice del driver, ma per completezza, nel nome identificante la misura, abbiamo inserito anche le unità di misura. Nella riga 13 è inserita l'istruzione Python deputata a creare la classe tramite cui il software DSlogger creerà l' oggetto (in senso informatico) che rappresenterà il dispositivo nel sistema. Il nome attribuito alla classe soddisfa il requisito espresso nel punto 3). In riga 15 troviamo il metodo costruttore della classe Pms5003 dove tutte le variabili locali necessarie al funzionamento vengono inizializzate ai valori opportuni. Nelle righe 24, 27, 33, 36, 39, 54 e 93 troviamo l'implementazione dei metodi pubblici obbligatori come descritto nel punto 4), mentre gli altri due metodi sono opzionali e implementati a descrizione dello scrittore del codice. Un elemento importante da sottolineare è che, una volta stabilito quante misure il dispositivo fornisce in output, congruentemente, il metodo "*sample(self)*" dovrà restituire in uscita sempre lo stesso numero di misure. Il numero di misure in output è deciso impostando la variabile *DEVICE_SENSORS*. Se essa è una stringa con due caratteri ";" al suo interno (ad esempio: "pm1[ug/m3];pm2.5[ug/m3];pm10[ug/m3]"), ciò vorrà indicare che il dispositivo fornisce in output 3 misure diverse. Di conseguenza, il metodo "*sample(self)*" ritornerà sempre una stringa con due caratteri";" al suo interno che delimitano i valori delle misure lette dal dispositivo.

Un ultima considerazione è necessaria sul metodo obbligatorio "*connect(self,serport)*", nel caso di dipsositivi su USB o RS232, o "*connect(self,address)*", nel caso di dispositivi "ethernet". Tale metodo è utilizzato dal sistema per verificare se un dato dispositivo è connesso e attivo su una determinata interfaccia hardware. Nell'implementare tale metodo, si invia un comando specifico del dispositivo dal quale si attende una risposta nota a priori. Se tale risposta sarà rilevata, allora significherà che il dispositivo è effettivamente connesso e attivo. Negli allegati è possibile visionare il codice relativo ad un driver scritto per l'analizzatore chimico AC32 prodotto dalla ENVEA che si interfaccia tramite porta ethernet. Valgono le stesse considerazioni già illustrate per il driver del sensore PMS5003 con le opportune varianti evidenziate nelle tabelle 4 e 5 che differenziano dispositivi che si interfacciano su seriale e Ethernet.

Al fine di riassumere tutte le informazioni fin qui esposte circa la scrittura dei driver dei dispositivi e, contemporaneamente, per offrire un comodo riferimento nella stesura dei codici relativi, sono esposti negli allegati i flles modello (templates) utilizzabili come guida per nuovi drivers di dispositivi seriali e ethernet.

6. Test funzionali

Diversi test sono stati effettuati per verificare la piena funzionalità del software attualmente implementato nelle versioni DSlogger1.0, utilizzabile con macchine a 32 bit, e DSlogger2.0, utilizzabile con macchine più recenti a 64 bit. Le prove effettuate hanno tutte avuto esito positivo. Per i test relativi alla prima versione è stato utilizzato un PC a 32 bit sul quale è installato un sistema operativo Windows XP con service pack 2. Per quanto riguarda invece i test relativi alla seconda versione, essi sono stati effettuati con PC a 64 bit su cui è installato il sistema operativo Windows 11.

Tutti i dispositivi elencati nella tabella 1 sono stati sottoposti a diverse prove, ma in questa sede, per brevità, riportiamo i dati relativi alla prova generale effettuata su Windows 11 nella quale sono stati utilizzati i sensori MHZ19 e PMS5003, e gli strumenti GO3, 405nm, AC32, CO12M, e O342. In tale prova, le misurazioni con i suddetti dispositivi sono state effettuate in ambiente indoor, per la precisione, nel laboratorio test sensori. Le misurazioni effettuate sono riassunte nelle figure 11, 12, 13, 14 e 15 che illustrano i dati raggruppandoli per omogeneità di misura effettuata. I grafici riprodotti nelle figure sono stati ricavati dall'unico file CSV prodotto da DSlogger2.0 utilizzando l'applicazione Origin7 della

Originlab, tuttavia, è possibile utilizzare una qualsiasi applicazione in grado di leggere files CSV e di graficarli, come ad esempio, Microsoft Excel.

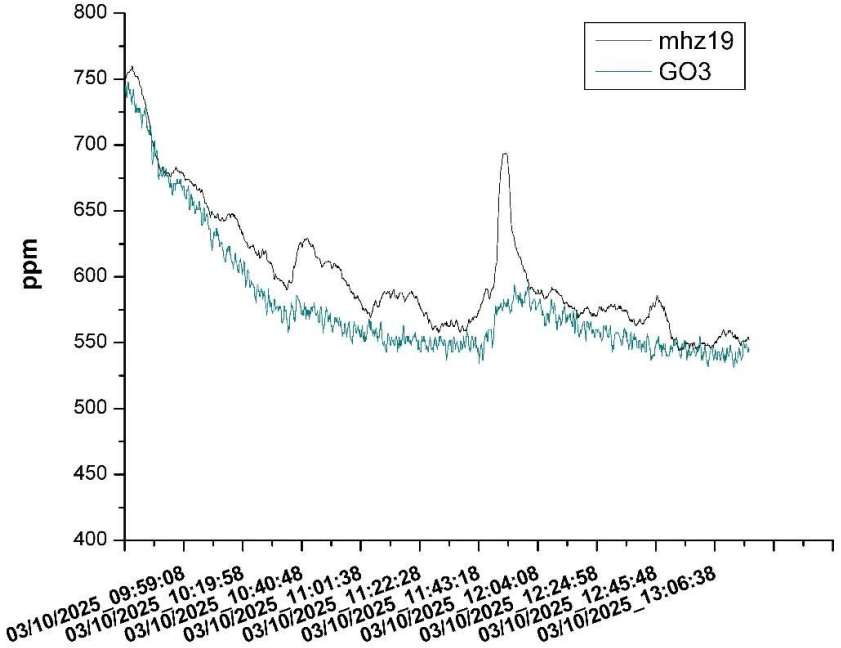

Figura 11: le misurazioni di CO_2 effettuate dal sensore MHZ19 e dallo strumento GO3

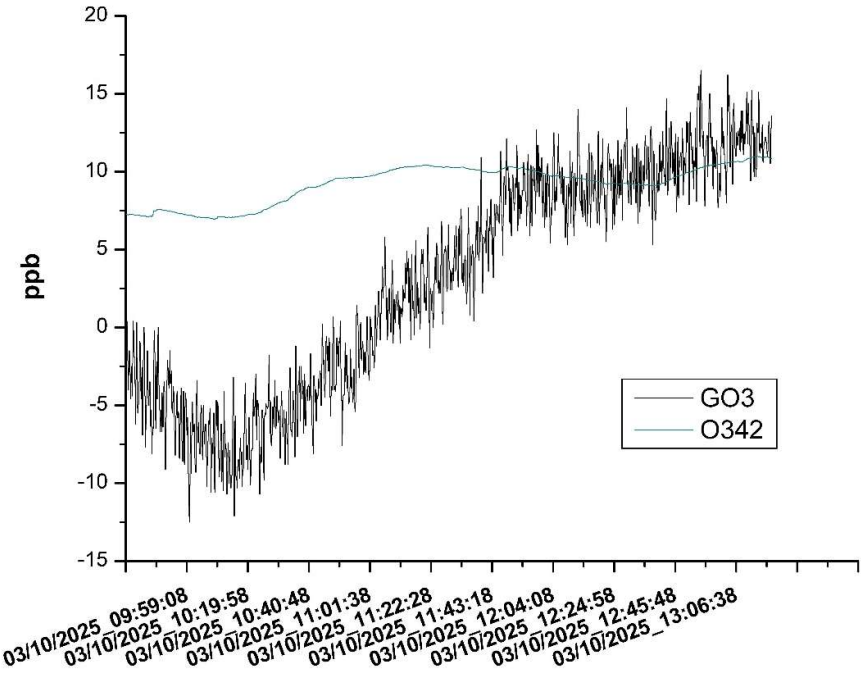

Figura 12: le misurazioni di ozono effettuate dallo strumento GO3 e dall' analizzatore chimico O342.

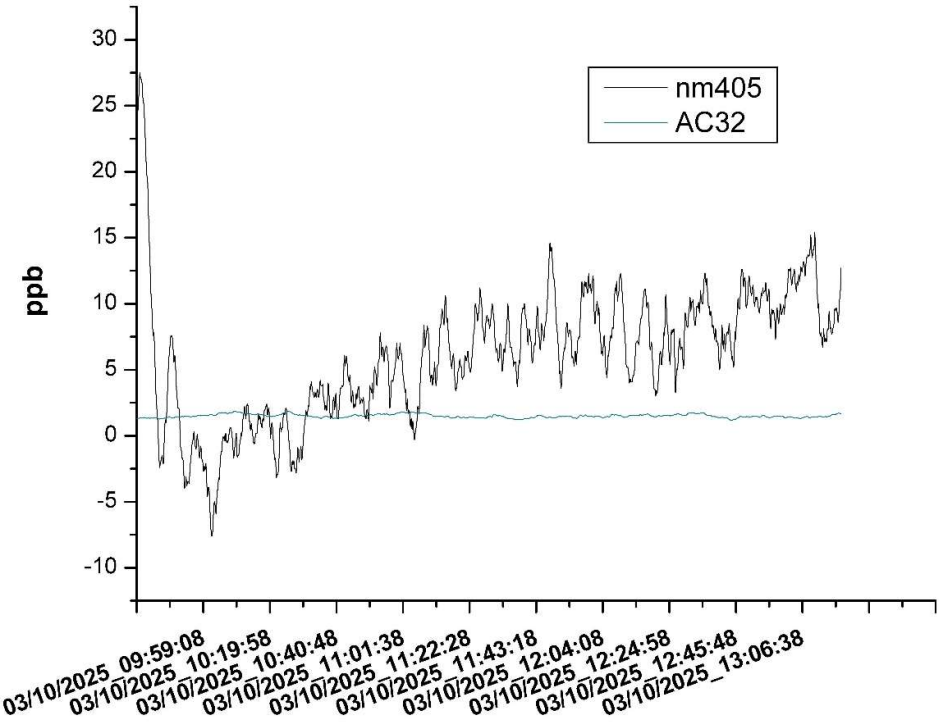

Figura 13: le misurazioni di NO_2 effettuate dallo strumento nm405 e dall'analizzatore chimico AC32.

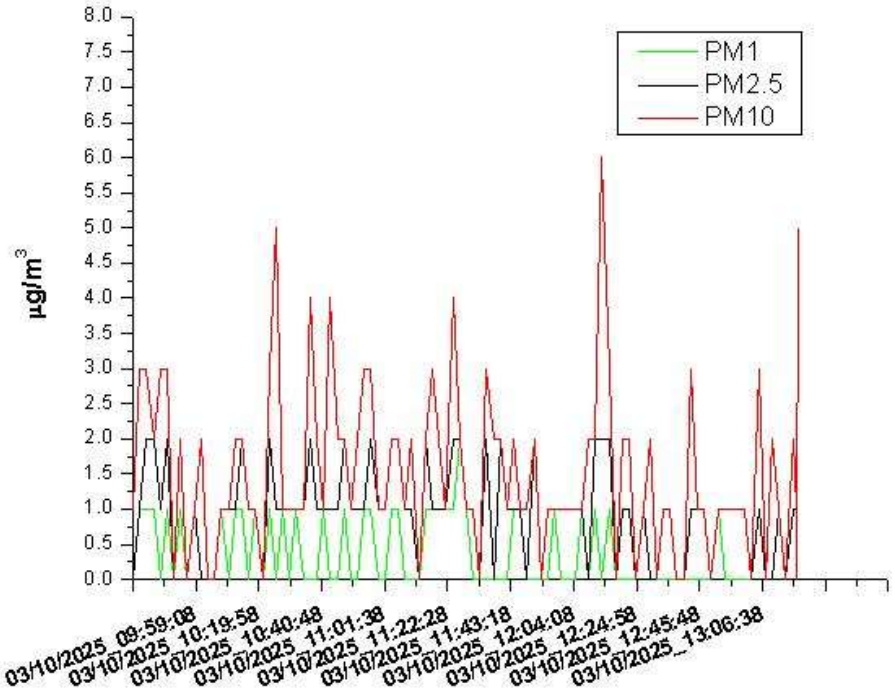

Figura 14: le misurazioni di polveri sottili effettuate dal sensore PMS5003.

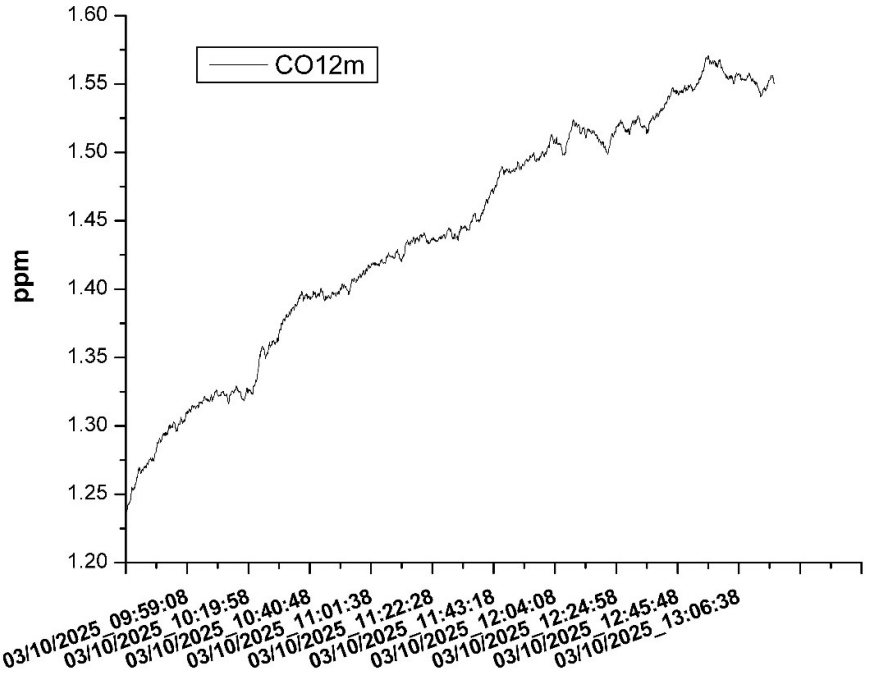

Figura 15: le misurazioni di CO effettuate dall'analizzatore chimico CO12m.

In figura 11 è riportata la concentrazione di anidride carbonica rilevate dal sensore e dallo strumento impiegato nel test. Ricordando che tale test è stato effettuato campionando l'aria presente nel laboratorio in presenza di personale, i livelli di anidride carbonica rilavati sono compatibili con un ambiente indoor occupato da una o più persone e, pertanto, sono ben maggiori della media attuale riscontrabile in ambienti esterni (pari a circa 400 ppm). Il picco evidenziato dal sensore mhz19 in posizione centrale al grafico è probabilmente dovuto alla presenza ravvicinata di personale al sensore. Le figure 12 e 13 mostrano invece diversi livelli di accuratezza degli strumenti coinvolti nel test e diversi livelli di "rumore" caratterizzante gli output dei dispositvi. Per quanto concerne la figura 14, dobbiamo considerare che il sensore PMS5003 ha una risoluzione di 1 ug/m^3. Tale dato, insieme alla bassa concentrazione di polveri sottili misurata, spiegano l'andamento quasi a gradini del grafico prodotto. Analizzando la figura 15, possiamo osservare che i livelli di monossido di carbonio sono ampiamente inferiori al limite di 10 ug/m^3 mediato su 8 ore con media mobile. Anche in questo caso, i valori rilevati sono compatibili con quelli che si possono tipicamente riscontrare in ambiente indoor.

7. Conclusioni

Uno strumento software, denominato DSlogger, è stato progettato e implementato per permettere la raccolta di dati provenienti di dispositivi di tipo eterogeneo. Esso consente un immediata visualizzazione di tutte le misure in tempo reale e la creazione di un file CSV, che funge da database dei dati e rappresenta l'output di tale strumento. Tale software risulta particolarmente utile in laboratorio, in stazioni di monitoraggio ove sono presenti strumenti e dispositivi di varia natura, o in tutte quelle situazioni nelle quali sia necessario acquisire diverse misure provenienti da fonti eterogenee.

Tra i suoi punti di forza, possiamo annoverare senza dubbio la sua economicità, essendo realizzato in ambiente software "open-source". Inoltre, un altro possibile vantaggio derivante da un suo utilizzo è

rappresentato dal fatto che esso permette di utilizzare vecchi PC obsoleti, ma pienamente funzionanti, permettendone un loro riciclo o virtuoso riutilizzo. Possibili sviluppi futuri riguardano l' investigazione sulla possibilità di scrittura di nuovi driver di dispositivi utilizzando strumenti di intelligenza artificiale, già attualmente disponibili, capaci di generare codice Python. Ciò consentirebbe anche a personale non esperto in programmazione python di poter creare autonomamente nuovi driver per i propri dispositivi.

Riferimenti

[1] C. -C. Chang, Y. -T. Chiu and C. -C. Wei, "Design instrument control software interface based on SCPI commands to reduce development time," 2021 7th International Conference on Applied System Innovation (ICASI), Chiayi, Taiwan, 2021, pp. 97-100, doi: 10.1109/ICASI52993.2021.9568467.

[2] Sito web: https://www.ivifoundation.org/About-IVI/scpi.html (ultimo accesso: 26 gennaio 2026).

[3] Sito web: https://www.ni.com/docs/en-US/bundle/labview/page/using-visa-in-labview.html (ultimo accesso: 26 gennaio 2026).

[4] Sito web: https://it.mathworks.com/products/instrument.html (ultimo accesso: 26 gennaio 2026).

[5] Perkel, J. Programming: Pick up Python. Nature 518, 125–126 (2015). https://doi.org/10.1038/518125a

[6] Sito web: https://www.alessiopomaro.it/chatgpt-python-analisi-dati-ai (ultimo accesso: 26 gennaio 2026).

[7] Sito web: https://chatgpt.com/ (ultimo accesso: 26 gennaio 2026).

[8] Sito web: https://gemini.google.com/app (ultimo accesso: 26 gennaio 2026).

[9] Sito web: https://culturedigitali.eu/notizie/esempi-di-codice-con-gemini-ai-python-javascript-e-altro/ (ultimo accesso: 26 gennaio 2026).

[10] Pagina web del prodotto SPS30: SPS30 - PM2.5 Sensor for HVAC and air quality applications SPS30 (ultimo accesso: 26 gennaio 2026).

[11] Pagina web del prodotto PMS5003: https://www.plantower.com/en/products_33/74.html (ultimo accesso: 26 gennaio 2026).

[12] Pagina web del prodotto 405nm: https://2btech.io/items/other-monitors/model-405-nm-no2-no-nox-monitor/ (ultimo accesso: 26 gennaio 2026).

[13] Pagina web del prodotto MH-Z19: https://www.winsen-sensor.com/product/mh-z19e.html (ultimo accesso: 26 gennaio 2026).

[14] Pagina web del prodotto IRC-A1: https://www.alphasense.com/-/media/project/oneweb/oneweb/alphasense/products/datasheets/alphasense_ndir-transmitter.pdf (ultimo accesso: 26 gennaio 2026).

[15] Jessa A. Ellenburg, Craig J. Williford, Shannon L. Rodriguez, Peter C. Andersen, Andrew A. Turnipseed, Christine A. Ennis, Kali A. Basman, Jessica M. Hatz, Jason C. Prince, Drew H. Meyers, David J. Kopala, Michael J. Samon, Kodi J. Jaspers, Boden J. Lanham, Brian J. Carpenter, John W. Birks, Global Ozone (GO3) Project and AQTreks: Use of evolving technologies by students and citizen

scientists to monitor air pollutants, Atmospheric Environment: X, Volume 4, 2019, 100048, https://doi.org/10.1016/j.aeaoa.2019.100048

[16] Pagina web del prodotto V72M: https://envea.global/design/pdf/envea_voc72m_gas-chromatography-voc-compounds_en.pdf (ultimo accesso: 26 gennaio 2026).

[17] Pagina web del prodotto O342: https://envea.global/it/product/o342e/ (ultimo accesso: 26 gennaio 2026).

[18] Pagina web del prodotto CO12M: https://envea.global/it/product/co12e/ (ultimo accesso: 26 gennaio 2026).

[19] Pagina web del prodotto AF22: https://envea.global/it/product/af22e/ (ultimo accesso: 26 gennaio 2026).

[20] Pagina web del prodotto AC32: https://envea.global/it/product/ac32e-2/ (ultimo accesso: 26 gennaio 2026).

[21] Pagina web dell' "hub" ethernet LS105G prodotto dalla TP-link: https://www.tp-link.com/it/home-networking/soho-switch/ls105g/ (ultimo accesso: 26 gennaio 2026).

[22] Pagina web dell' adattatore RS232/USB UT232R-500 prodotto dalla FTDI chip: https://ftdichip.com/products/ut232r-500/ (ultimo accesso: 26 gennaio 2026).

[23] Pagina web dell' adattatore RS232/USB UT232R-500 prodotto dalla FTDI chip: https://ftdichip.com/products/usb-rs232-pcb/ (ultimo accesso: 26 gennaio 2026).

[24] Pagina web da dove è possibile scaricare "DSlogger1.0": https://eneait-my.sharepoint.com/:f:/g/personal/domenico_suriano_enea_it/IgCEzK8iEoXYQKFu3mbYfvSWAQZTqCOgLxYbyXWwMZMbavA?e=QmGbGV (ultimo accesso: 26 gennaio 2026).

[25] Pagina web da dove è possibile scaricare "DSlogger2.0": https://eneait-my.sharepoint.com/:f:/g/personal/domenico_suriano_enea_it/IgBYIx2-EJQYRY6lVqvGFMAoAa_bWIZJTjxG3EmmbV4UGQ0?e=peOxDg (ultimo accesso: 26 gennaio 2026).

Allegati

Codice contenuto nel file "DSlogger.py"

```
installed_devices = []

import serial
import serial.tools.list_ports
import time
import os
import socket
import copy
import sys
import _thread
import importlib
import signal

DEVICE_DIR = "devices"
DATA_DIR = "data"
INST_INIT = "installed.ini"
TEMP_FILE = "__temp.tmp"
```

```
USB_CONNECTION_TYPE = "usb"
SERIAL_CONNECTION_TYPE = "serial"
ETH_CONNECTION_TYPE = "eth"

VIEWER_ADDRESS = ('localhost', 1000)
MINIMUM_RATE = 5

global rate

#connected devices are stored here
connected_devices = []


def help():
    print("\npress h[ENTER] for command viewing")
    print("press s[ENTER] for searching devices")
    print("press c[ENTER] for checking devices")
    print("press b[ENTER] for stopping sampling sessions")
    print("press q[ENTER] to quit the app")
    print("type s,<sampling rate in seconds>[ENTER] to start and log a
sampling session")
    print("type i,<new_device_module_name>[ENTER] to install a new device
driver")
    print("type u,<device_name_to_uninstall>[ENTER] to uninstall a new
device driver")


## devices scanning: this routine search devices and the ports where they
are plugged into.
## Then it creates the connections
def device_scanning(conn_dev,dev):
    print("\nSearching for devices connected...")
    for cn in conn_dev:
        cn.terminate()
        del cn
    conn_dev = []
    ports = list(serial.tools.list_ports.comports())
    for prt in ports:
        for dv in dev:
            conn_type = dv.getConnectionType()
            if (conn_type == USB_CONNECTION_TYPE) or (conn_type ==
SERIAL_CONNECTION_TYPE):
                print ("\nSearching for " + dv.getIdentity() + " on " +
prt[0] + " port")
                conn_dev.append(copy.deepcopy(dv))
                if conn_dev[-1].connect(prt[0]) == 1:
                    sens = conn_dev[-1].getSensors()
                    meas = conn_dev[-1].sample()
                    num_sens = sens.split(';')
                    num_meas = meas.split(';')
                    if len(num_sens) != len(num_meas):
                        conn_dev[-1].terminate()
                        del conn_dev[-1]
                        print (dv.getIdentity() + " NOT FOUND")
                        continue
                    print ("FOUND " + conn_dev[-1].getIdentity())
                    print ("measures: " + conn_dev[-1].getSensors())
                    #updating device identity for multi-copies purposes
                    original_identity = conn_dev[-1].getIdentity()
```

```
                    conn_dev[-1].setIdentity(original_identity + "-" +
prt[0])
                    break
                else:
                    print (dv.getIdentity() + " NOT FOUND")
                    del conn_dev[-1]
            else:
                continue
    for dv in dev:
        conn_type = dv.getConnectionType()
        if conn_type == ETH_CONNECTION_TYPE:
            conn_par = dv.getConnectionParams()
            for address in conn_par:
                print ("\nSearching for " + dv.getIdentity() + " on " +
address)
                conn_dev.append(copy.deepcopy(dv))
                if conn_dev[-1].connect(address) == 1:
                    sens = conn_dev[-1].getSensors()
                    meas = conn_dev[-1].sample()
                    num_sens = sens.split(';')
                    num_meas = meas.split(';')
                    if len(num_sens) != len(num_meas):
                        conn_dev[-1].terminate()
                        del conn_dev[-1]
                        print (dv.getIdentity() + " NOT FOUND")
                        continue
                    print ("FOUND " + conn_dev[-1].getIdentity())
                    print ("measures: " + conn_dev[-1].getSensors())
                else:
                    print (dv.getIdentity() + " NOT FOUND")
                    del conn_dev[-1]
    if len(conn_dev) == 0:
        print("\nNo device found!")
    else:
        print("\nDevices found:")
        for cd in conn_dev:
            print(cd.getIdentity())
    return conn_dev


def capture(sk1,conn_dvc,appdir):
    global rate
    str_record,str_log = make_record(conn_dvc)
    sent = sk1.sendto(str_record.encode(), VIEWER_ADDRESS)
    cur_datafile = logging_init(conn_dvc,appdir)
    res = log_measures(str_log,cur_datafile)
    adesso = time.time()
    while(rate!=0):## loop where devices are read if it is the time
        dopo = time.time()
        diff = int(dopo-adesso)
        if(diff>=rate):## now it is the time to read the devices, gather and
treat the data
            adesso = time.time()
            if rate != 0:
                str_record1,str_log1 = make_record(conn_dvc)
                sent1 = sk1.sendto(str_record1.encode(), VIEWER_ADDRESS)
                res = log_measures(str_log1,cur_datafile)
```

```
## builds the record to store in the file data by gathering measure from all
the devices connected
def make_record(conn_dvc):
    strtosend = time.strftime("%d/%m/%Y_%H:%M:%S")
    strtolog = strtosend +  ";"
    for cnd in conn_dvc:
        meas = cnd.sample()
        strtosend = strtosend + ' ' + cnd.getIdentity() + ' ' +
cnd.getSensors() + ' ' + meas
        strtolog = strtolog + meas + ";"
    strtolog1 = strtolog.rstrip(";")
    return strtosend,strtolog1


def check_devices(conn_dev,dev):
    print("The following devices are now installed on the system:")
    try:
        mp1 = INST_INIT
        f = open(mp1,"r")
        contents = f.readlines()
        f.close()
    except Exception as e:
        print ("no device installed")
        return
    for line in contents:
        if ((line == "\n") or (line == "\r") or (line == "\r\n")):
            continue
        dev1 = line.replace('\r','')
        dev2 = dev1.replace('\n','')
        devdriver = dev2.replace('.py','')
        print(devdriver)
    print("\nThe devices currently connected are:")
    for cnd in conn_dev:
        print(cnd.getIdentity())
        print(cnd.getSensors())
        print(cnd.sample())
        print("")


def logging_init(conn_dvc,ad):
    temp = time.strftime("%Y-%m-%d_%H-%M-%S")
    dirdata = ad + "\\" + DATA_DIR
    if not os.path.exists(dirdata):
        os.makedirs(dirdata)
    headerdata = "Date/time;"
    for cnd in conn_dvc:
        dev_id = cnd.getIdentity()
        dev_meas = cnd.getSensors()
        meashead = dev_meas.split(";")
        for meas in meashead:
            headerdata = headerdata + dev_id + "_" + meas + ";"
    filedata = dirdata + "\\" + temp + ".txt"
    try:
        f = open(filedata, 'a+')
        f.write(headerdata.rstrip(";") + "\n")
        f.close()
        return filedata
    except Exception as e:
        return ""
```

```
def log_measures(rectolog,logfile):
    try:
        f = open(logfile, 'a')
        f.write(rectolog + "\n")
        f.close()
        return 0
    except Exception as e:
        return 1


### function to install new device drivers on the system
def install_device(devname,appdir):
    devdir = appdir + '\\' + DEVICE_DIR
    devname = devname.replace('\r','')
    devname = devname.replace('\n','')
    devname1 = devname.rstrip(".py")
    if os.path.exists(devdir + "\\" + devname1 + ".py")==False:
        print("Error: " + devname1 + ".py not found in " + devdir)
        print("Reminder: to install a new device driver, first")
        print("put the \'your_device_driver.py\' in the \\" + DEVICE_DIR + "
directory!")
        return
    if os.path.exists(devdir)==False:
        print("Error: " + DEVICE_DIR + " directory does not exist.")
        print("To install device drivers, first create")
        print("a directory called" + DEVICE_DIR + "in the program")
        print("directory, then put in it device driver files,")
        print("and finally proceed to install.")
        return
    else:
        try:
            f1 = INST_INIT
            f = open(f1,"r")
            contents = f.readlines()
            f.close()
            for line in contents:
                line1 = line.replace('\r','')
                line2 = line1.replace('\n','')
                line3 = line2.rstrip(".py")
                if devname1 == line3:
                    print("Attention: " + devname1 + " already installed on
the system!")
                    return
        except Exception as e:
            pass
        with open(INST_INIT, 'a') as file:
            file.write(devname1 +"\n")
            file.close()
            print(devname1 + " successfully installed.")
            print("Please, restart the app to load the new device.")


### function to uninstall device drivers in the system
def uninstall_device(devname,appdir):
    devdir = appdir + '\\' + DEVICE_DIR
    devname = devname.replace('\r','')
    devname = devname.replace('\n','')
    devname1 = devname.rstrip(".py")
    found = 0
```

```
    if os.path.exists(appdir + "\\" + INST_INIT) == False:
        print("No device to uninstall. \'" + INST_INIT + "\'")
        print("is not present in the app directory.")
        return
    else:
        try:
            f1 = INST_INIT
            f = open(f1,"r")
            contents = f.readlines()
            f.close()
            newcontent = []
        except Exception as e:
            print("Impossible to uninstall " + devname1 + ":")
            print(str(e))
            return
        for line in contents:
            if line.find(devname1) == -1:
                newcontent.append(line)
            else:
                found = 1
    with open(INST_INIT, 'w') as file:
        file.writelines(newcontent)
        file.close()
    if found == 1:
        print(devname1 + " successfully uninstalled.")
        print("Please, restart the app to implement the uninstallation.")
    else:
        print("No device named " + devname1 + " found to uninstall.")


def load_devices():
    print("\nLoaading device drivers...")
    found = 0
    try:
        mp1 = INST_INIT
        f = open(mp1,"r")
        contents = f.readlines()
        f.close()
    except Exception as e:
        print ("No device installed")
        return found
    for line in contents:
        if ((line == "\n") or (line == "\r") or (line == "\r\n")):
            continue
        dev1 = line.replace('\r','')
        dev2 = dev1.replace('\n','')
        devdriver = dev2.replace('.py','')
        dev_class = devdriver.capitalize()
        try:
            dev_module = importlib.import_module(devdriver)
            dev_class = getattr(dev_module,dev_class)
            dev_obj = dev_class()
            installed_devices.append(dev_obj)
            found = found + 1
            print("Loaded " + devdriver + ".py driver ")
        except Exception as e:
            print("Impossible to load " + devdriver + ".py driver: " +
str(e))
    if found == 0:
        print("No device correctly installed!")
```

```
    return found


def viewer_opening():
    str1 = "start data_viewer.exe " + str(VIEWER_ADDRESS[1])
    try:
        os.system(str1)
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        return sock
    except:
        print("unable to open viewer window: check if already opened\n\r")
        return None


######################
######### MAIN ########
######################
sk = viewer_opening()
os.system('cls')
curdir1 = os.getcwd()
sys.path.append(DEVICE_DIR)
print("Welcome on DSlogger 2.0 by Domenico Suriano!")
numdev = load_devices()
if numdev > 0:
    connected_devices = device_scanning(connected_devices,installed_devices)
print("\nDSlogger 2.0 by Domenico Suriano ready!")
help()
rate = 0

while 1:
    command = input(">>> ")
    if command == 's':
        if (rate == 0):
            if numdev > 0:
                connected_devices =
device_scanning(connected_devices,installed_devices)
        else:
            print("Impossible to execute the command:")
            print("sampling session ongoing,")
            print("if you want to search new devices")
            print("press 'b' to stop the session!\n\r")
    elif command == 'b':
        if rate != 0:
            rate = 0
            print("Logging stopped without errors")
    elif command == 'c':
        if rate == 0:
            check_devices(connected_devices,installed_devices)
        else:
            print("Impossible to execute the command:")
            print("sampling session ongoing,")
            print("if you want to check on devices")
            print("press 'b' to stop the session!")
    elif command == 'h':
        help()
    elif (command.find('i') == 0) and (len(command)>2) and command.find(',')
== 1:
        if rate == 0:
            devname = command.split(',')
            install_device(devname[1],curdir1)
```

```
        else:
            print("Logging ongoing. If you want to install new devices,")
            print("first stop the current logging by")
            print("typing the command \'b\'")
    elif (command.find('u') == 0) and (len(command)>2) and command.find(',')
== 1:
        if rate == 0:
            devname = command.split(',')
            uninstall_device(devname[1],curdir1)
        else:
            print("Logging ongoing. If you want to uninstall a device,")
            print("first stop the current logging by")
            print("typing the command \'b\'")
    elif (command.find('s') == 0) and (len(command)>2) and command.find(',')
== 1:
        if (len(connected_devices) > 0):
            if rate == 0:
                par = command.split(',')
                try:
                    rt = int(par[1])
                    rate = rt
                    print("Logging started at " + par[1] + " sec. sampling
rate")

_thread.start_new_thread(capture,(sk,connected_devices,curdir1))
                except Exception as e:
                    print(str(e))
                    strout = "ERROR: " + par[1] + " is not a valid number"
                    print(strout)
            else:
                print("Logging ongoing. If you want to log new
measurement,")
                print("first stop the current logging by")
                print("typing the command \'b\'")
        else:
            print("No device connected. Impossible to start logging!")
    elif command == 'q':
        try:
            f = open(TEMP_FILE, "r")
            pid = f.read()
            f.close()
            os.kill(int(pid),signal.SIGTERM)
            os.remove(TEMP_FILE)
        except Exception as e:
            pass
        sys.exit(0)
    else:
        print("Invalid command")
```

Codice contenuto nel file driver denominato “pms5003.py”relativo al dispositivo PMS5003.

```
import serial
import time

DEVICE_IDENTITY = "PMS5003"
CONNECTION_TYPE = "serial"
DEVICE_BAUD_RATE = 9600
DEVICE_SENSORS = "pm1[ug/m3];pm2.5[ug/m3];pm10[ug/m3]"

SERIAL_TIMEOUT = 2
MAX_NUM_ATTEMPT = 10
ERR_VAL = "-100"

class Pms5003:

    def __init__(self):
        self.identity = DEVICE_IDENTITY
        self.connection_type = CONNECTION_TYPE
        self.sensors = DEVICE_SENSORS
        self.baud_rate = DEVICE_BAUD_RATE
        self.portname = ""
        self.port = None
        self.strmeas = "0.0;0.0;0.0"

    def getSensors(self):
        return self.sensors

    def getIdentity(self):
        return self.identity

    def setIdentity(self,idstring):
        self.identity = idstring

    def getConnectionType(self):
        return self.connection_type

    def getConnectionParams(self):
        return [self.portname,self.baud_rate]

    def terminate(self):
        self.lesteningthread = 0
        try:
            self.port.close()
        except:
            return

    def __del__(self):
        try:
            self.port.close()
        except:
            return

## the function "connect" check if the device is plugged into serport,
## then returns 1 if the device is found
    def connect(self,serport):
        found = 0
        if (serport.find("ttyACM")>= 0):
            return found
        try:
            self.port = serial.Serial(serport,self.baud_rate, timeout =
SERIAL_TIMEOUT, rtscts=0)
```

```
61          except Exception as e:
62              return found
63          num_attempt = 0
64          while num_attempt < MAX_NUM_ATTEMPT:
65              try:
66                  time.sleep(0.3)
67                  self.port.write([66, 77, 225, 0, 0, 1, 112])
68                  buf = self.port.read(8)
69                  if (buf[0] == 66) and (buf[1] == 77):
70                      ck = 0
71                      for a in range(5):
72                         ck = ck + buf[a]
73                      ck = ck & 255
74                      if ck == buf[7]:
75                          self.portname = serport
76                          found = 1
77                          return found
78                  if len(buf) == 0:
79                      num_attempt = num_attempt + 1
80                      continue
81                  if (buf[0] != 66) or (buf[1] != 77):
82                      num_attempt = num_attempt + 1
83                      self.port.close()
84                      time.sleep(1)
85                      self.port.open()
86              except Exception as e:
87                  num_attempt = num_attempt + 1
88          return found
89
90  ## the function "sample" reads the sensor output,
91  ## then returns a string separate by semicolon containing data
92  ## if an error happens, it returns the string with err value: "-100.0"
93      def sample(self):
94          num_attempt = 0
95          self.strmeas = ERR_VAL + ";" + ERR_VAL + ";" + ERR_VAL
96          while num_attempt < MAX_NUM_ATTEMPT:
97              try:
98                  time.sleep(0.4)
99                  self.port.write([66, 77, 226, 0, 0, 1, 113])
100                 buf = self.port.read(32)
101                 if (buf[0] == 66) and (buf[1] == 77):
102                     cs = (buf[30] * 256 + buf[31])
103                     check = 0
104                     for i in range(30):
105                         check += buf[i]
106                     if check != cs:
107                         num_attempt = num_attempt + 1
108                         continue
109                     pm1 = float(buf[10] * 256 + buf[11])
110                     pm25 = float(buf[12] * 256 + buf[13])
111                     pm10 = float(buf[14] * 256 + buf[15])
112                     self.strmeas = str(pm1) + ";" + str(pm25) + ";" +
113 str(pm10)
114                     return self.strmeas
115             except Exception as e:
116                 num_attempt = num_attempt + 1
117         return self.strmeas
```

Codice contenuto nel file driver denominato "ac32.py"relativo al dispositivo AC32.

```
import socket
import time

CONNECTION_TYPE = "eth"
ETH_ADDRESSES = ["192.168.20.43"]
DEVICE_IDENTITY = "AC32"
DEVICE_SENSORS = "no[ppb];nox[ppb];no2[ppb]"

AC32_PORT = 8000
SOCKET_TIMEOUT = 2
MAX_NUM_ATTEMPT = 4
ACK = '\x06'
ERR_VAL = "-100"

class Ac32:

    def __init__(self):
        self.identity = DEVICE_IDENTITY
        self.connection_type = CONNECTION_TYPE
        self.sensors = DEVICE_SENSORS
        self.addresses = ETH_ADDRESSES
        self.address = None
        self.sk = None
        self.port = AC32_PORT
        self.status = 'U'
        self.alarm = None

## the function "connect" check if the device is plugged into Ethernet
port,
## then returns 1 if the device is found
    def connect(self,address):
        found = 0
        try:
            self.sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            self.sk.settimeout(SOCKET_TIMEOUT)
        except Exception as e:
            return found
        command = chr(2) + "AC32" + "16"
        tocalc = command.encode()
        c1,c2 = self.BCCcalc(tocalc)
        command = command + chr(c1) + chr(c2) + chr(3)
        tent = 0
        while tent < MAX_NUM_ATTEMPT:
            try:
                sent = self.sk.sendto(command.encode(),
(address,self.port))
                datal, server = self.sk.recvfrom(1024)
                data = datal.decode()
            except socket.timeout:
                if tent >= MAX_NUM_ATTEMPT:
                    return found
                else:
                    tent = tent + 1
                    time.sleep(0.2)
                    continue
            except Exception as e:
                if tent >= MAX_NUM_ATTEMPT:
                    return found
                else:
                    tent = tent + 1
```

```
                    time.sleep(0.2)
                    continue
            if (data[0]!= ACK) and (tent>=MAX_NUM_ATTEMPT):
                return 4
            if (data[0]!= ACK) and (tent<MAX_NUM_ATTEMPT):
                tent = tent + 1
                time.sleep(0.2)
                continue
            if (data[0]== ACK):
                self.identity = str(data[1:5])
                self.status = data[13]
                self.alarm = str(data[14]) + str(data[15])
                self.address = address
                found = 1
                return found

    def getConnectionParams(self):
        return self.addresses

    def getConnectionType(self):
        return self.connection_type

    def getIdentity(self):
        return self.identity

    def setIdentity(self,idstring):
        self.identity = idstring

    def getSensors(self):
        return self.sensors

    def terminate(self):
        try:
            self.sk.close()
        except:
            return

    def __del__(self):
        try:
            self.sk.close()
        except:
            return


    def BCCcalc(self,mb):
        BCC = 0
        for i in range(1,len(mb)):
            BCC = BCC ^ mb[i]
        BCC1 = BCC>>4|48
        if BCC1 > 57:
            BCC1+=7
        BCC2 = (BCC&15)|48
        if BCC2 > 57:
            BCC2+=7
        return(BCC1,BCC2)

## the function "sample" reads the sensor output,
## then returns a string separate by semicolon containing data
## if an error happens, it returns the string with err value: "-100.0"
    def sample(self):
```

```
121             command = chr(2) + "AC32" + "16"
122             tocalc = command.encode()
123             c1,c2 = self.BCCcalc(tocalc)
124             command = command + chr(c1) + chr(c2) + chr(3)
125             tent = 0
126             while tent < MAX_NUM_ATTEMPT:
127                 try:
128                     sent = self.sk.sendto(command.encode(),
129     (self.address,self.port))
130                     data1, server = self.sk.recvfrom(1024)
131                     data = data1.decode()
132                 except socket.timeout:
133                     tent = tent + 1
134                     time.sleep(0.2)
135                     continue
136                 except Exception as e:
137                     tent = tent + 1
138                     time.sleep(0.2)
139                     continue
140                 if data[0]!= ACK:
141                     tent = tent + 1
142                     time.sleep(0.2)
143                     continue
144                 else:
145                     self.status = data[13]
146                     self.alarm = str(data[14]) + str(data[15])
147                     str2 = data[17:]
148                     strm = str2.split(' ')
149                     try:
150                         val1 = float(strm[0])
151                         val2 = float(strm[0])
152                         val3 = float(strm[0])
153                     except:
154                         return ERR_VAL + ";" + ERR_VAL + ";" + ERR_VAL
155                     if val1 < 0:
156                         val1str = "0.0"
157                     else:
158                         val1str = strm[0]
159                     if val2 < 0:
160                         val2str = "0.0"
161                     else:
162                         val2str = strm[1]
163                     if val3 < 0:
164                         val3str = "0.0"
165                     else:
166                         val3str = strm[2]
167                     return val1str + ';' + val2str + ';' + val3str
168             return ERR_VAL + ";" + ERR_VAL + ";" + ERR_VAL
169
```

Codice del file template per dispositivi con interfaccia di output di tipo seriale (USB, RS232 o UART)

```
# Copyright 2020   Dr. Domenico Suriano (domenico.suriano@enea.it)
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.


#########In this section you should import the necessary libraries.
import serial # mandatory library to import
#################################


##############In this section the necessary constants should be placed.
Here below it is an example
CONNECTION_TYPE = "serial" # mandatory string which indicates that the
device is interfaced by a serial port. Do not modify this one.

DEVICE_IDENTITY = "device name chosen by the user" # mandatory string
indicating the device name. Modify it with your device name.

# mandatory string indicating the magnitudes measured by the device. It
must have fields separated by ";"
# as in the example below
# DEVICE_SENSORS = "NO2[ppb];NO[ppb];NOx[ppb]". Modify it for your device
operation.
"""
if you have just one field, then the constant will be:
DEVICE_SENSORS = "NO2[ppb]"
"""
DEVICE_SENSORS = "field1[unit of measure 1];field2[unit of measure
2];field3[unit of measure 3]"

DEVICE_BAUD_RATE = your_device_baud_rate # mandatory integer value
indicating the serial communication baud rate
""" for example:
DEVICE_BAUD_RATE = 9600
"""
###################################

#### Here below you can put other constants specific for the operation of
your device
""" for example:
SERIAL_TIMEOUT = 2
"""
################################################


class Serial-device-template: #it is mandatory that the class name must
have the same name of the file (serial-device-template.py) with the
capital first letter

```

```
    def __init__(self):
        ### mandatory variables. Do not modify or cancel them.
        self.identity = DEVICE_IDENTITY # mandatory string setting device
identity
        self.connection_type = CONNECTION_TYPE # mandatory string setting
device output interface type
        self.sensors = DEVICE_SENSORS # mandatory string indicating the
magnitudes measured by the device
        self.baud_rate = DEVICE_BAUD_RATE # mandatory variable setting
the baud rate of your device
        self.portname = "" # mandatory string indicating the usb port
name temporarily set to Null.
        self.port = None # mandatory variable where it is going to be
stored the "serial" object
        ### here below you can put other variables specific for the
operation of your device

        ####################################################


### mandatory functions. Do not modify or cancel them.
    def getConnectionType(self):
        return self.connection_type

    def getConnectionParams(self):
        return [self.portname,self.baud_rate]

    def getIdentity(self):
        return self.identity

    def setIdentity(self,idstring):
        self.identity = idstring

    def getSensors(self):
        return self.sensors

    def terminate(self):
        try:
            self.port.close()
        except:
            return

    def __del__(self):
        try:
            self.port.close()
        except:
            return
############################################################


## the mandatory function "connect" check if your device is plugged into
the SentinAir serial port.
## It returns 1 if your device is found, 0 if it is not found.
## The mandatory argument to pass to the function is the port name (which
is a string
## that is going to be passed by the SentinAir system manager)
## where the function is going to search your device

    def connect(self,portname):
        found = 0
```

```
121          try:
122              ### put here the code specific for your device
123              """ for example:
124              self.port = serial.Serial(portname,self.baud_rate, timeout =
125  SERIAL_TIMEOUT, rtscts=0)
126              """
127              ################################
128              self.portname = portname ## if your device is found at the
129  passed serial port, the device serial port name is set.
130              found = 1
131          except Exception as e:
132              pass
133          return found
134
135
136  ## the mandatory function sample returns the reading of the magnitudes
137  measured by your device.
138  ## It is mandatory that the measures returned are in a string having
139  fields separated by ";". For example: "2.12;4.32;1.21;0.34"
140  ## The number of fields must be the same of the DEVICE_SENSORS constant.
141  If you have just one field,
142  ## the returned string will be,for example, "2.12"
143
144      def sample(self):
145          measures = ""
146          try:
147               ### put here the code specific for your device
148
149              ################################
150              return measures.rstrip(";")
151          except Exception as e:
152              return ""
153
154  ## Here below you can put other functions specific for your device
155  operation
156
157  ###################################################
```

Codice del file template per dispositivi con interfaccia di output di tipo ethernet

```
# Copyright 2020   Dr. Domenico Suriano (domenico.suriano@enea.it)
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.


#########In this section you should import the necessary libraries.
import socket # mandatory library to import
#################################


#############In this section the necessary constants should be placed.
Here below it is an example
CONNECTION_TYPE = "eth" # mandatory string which indicates that the
device is interfaced by the Ethernet socket. Do not modify this one.

# mandatory list which indicates the possible addresses where your device
must be searched.
# You can modify it by inserting the values right for your device
operation
# following the example below
# ETH_ADDRESSES = ["192.168.20.43","192.168.20.44"]
ETH_ADDRESSES = ["address1","address2"]
################################################################################
##############################

DEVICE_IDENTITY = "device name chosen by the user" # mandatory string
indicating the device name. Modify it with your device name.

# mandatory string indicating the magnitudes measured by the device. It
must have fields separated by ";"
# as in the example below
# DEVICE_SENSORS = "NO2[ppb];NO[ppb];NOx[ppb]". Modify it for your device
operation.
"""
if you have just one field, then the constant will be:
DEVICE_SENSORS = "NO2[ppb]"
"""
DEVICE_SENSORS = "field1[unit of measure 1];field2[unit of measure
2];field3[unit of measure 3]"
###################################

#### Here below you can put other constants specific for the operation of
your device
""" for example:
SOCKET_TIMEOUT = 2
"""
################################################


```

```
class Ethernet-device-template: #it is mandatory that the class name must
have the same name of the file (ethernet-device-template.py) with the
capital first letter

    def __init__(self):
        ### mandatory variables. Do not modify or cancel them.
        self.identity = DEVICE_IDENTITY # mandatory string setting device
identity
        self.connection_type = CONNECTION_TYPE # mandatory string setting
device output interface type
        self.sensors = DEVICE_SENSORS # mandatory string indicating the
magnitudes measured by the device
        self.addresses = ETH_ADDRESSES # mandatory list indicating the
possible addresses where it is possible to find your device on the net
        self.device_address = None # mandatory variable where it is going
to be stored your device address
        self.sk = None # mandatory variable which is going to store the
socket object from the "socket" library
        ### here below you can put other variables specific for the
operation of your device
        """ for example:
        self.port = 8000
        """
        ################################################


### mandatory functions. Do not modify or cancel them.
    def getConnectionType(self):
        return self.connection_type

    def getConnectionParams(self):
        return self.addresses

    def getIdentity(self):
        return self.identity

    def setIdentity(self,idstring):
        self.identity = idstring

    def getSensors(self):
        return self.sensors

    def terminate(self):
        try:
            self.sk.close()
        except:
            return

    def __del__(self):
        try:
            self.sk.close()
        except:
            return
############################################################


## the mandatory function "connect" check if your device is conected to
the net set up by SentinAir.
## It returns 1 if your device is found, 0 if it is not found.
```

```
## The mandatory argument to pass to the function is the address (which
is a string: e.g: "192.168.20.43")
## where the function is going to search your device
    def connect(self,address):
        found = 0
        try:
            ### put here the code specific for your device
            """
            for example:
            self.sk = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            self.sk.settimeout(SOCKET_TIMEOUT)
            data_sent = self.sk.sendto(command.encode(),
(address,self.port))
            etc...
            """
            ################################
            self.device_address = address ## if your device is found at
the passed address, the device address is set.
            found = 1
        except Exception as e:
            pass
        return found


## the mandatory function sample returns the reading of the magnitudes
measured by your device.
## It is mandatory that the measures returned are in a string having
fields separated by ";". For example: "2.12;4.32;1.21;0.34"
## The number of fields must be the same of the DEVICE_SENSORS constant.
If you have just one field,
## the returned string will be,for example, "2.12"
    def sample(self):
        measures = ""
        try:
             ### put here the code specific for your device

            ################################
            return measures.rstrip(";")
        except Exception as e:
            return ""

## Here below you can put other functions specific for your device
operation

#####################################################
```

ENEA
Servizio Promozione e Comunicazione
www.enea.it

Stampa: Laboratorio Tecnografico ENEA - Centro Ricerche Frascati

marzo 2026