Nota generale sull'RTE (Real Time Engine) e sulla visualizzazione delle pagine grafiche nell'ecosistema Syel + Codesys

13-01-2023

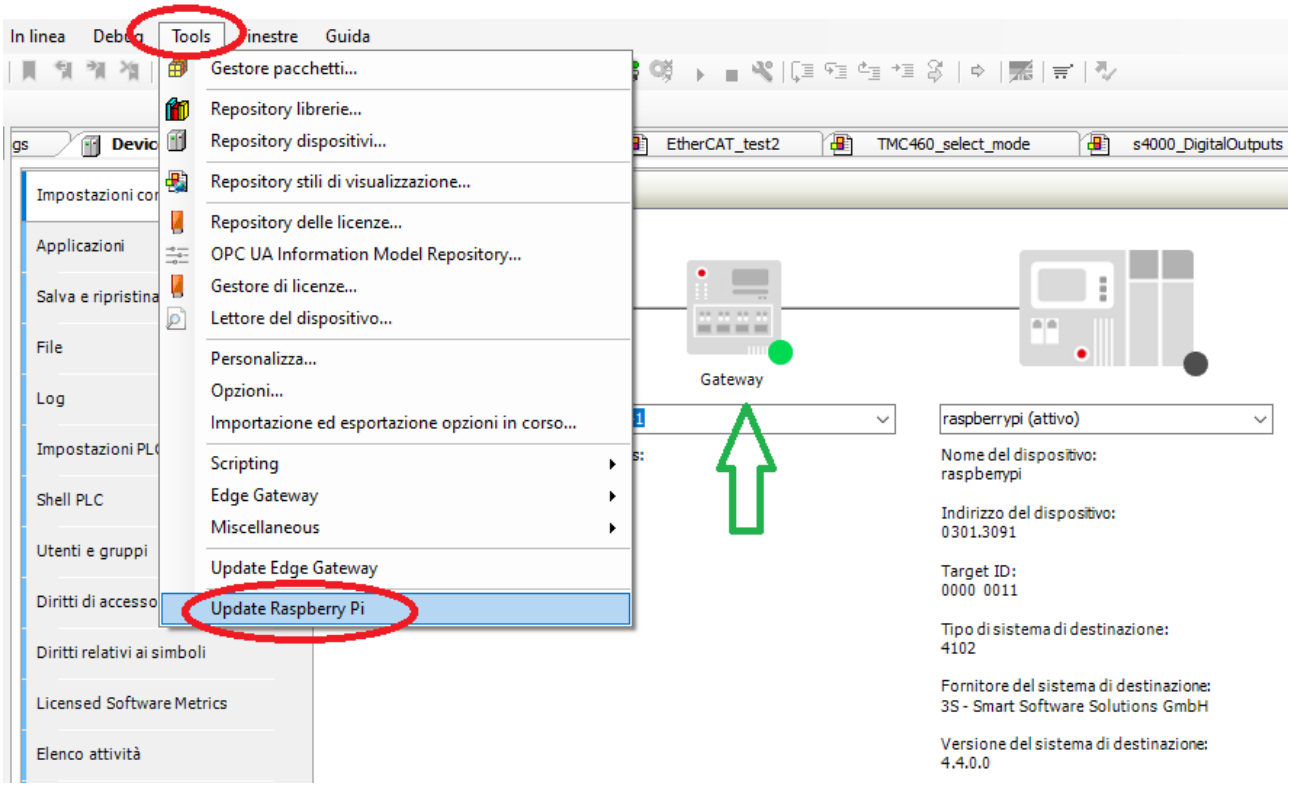
Draft, version 1.0

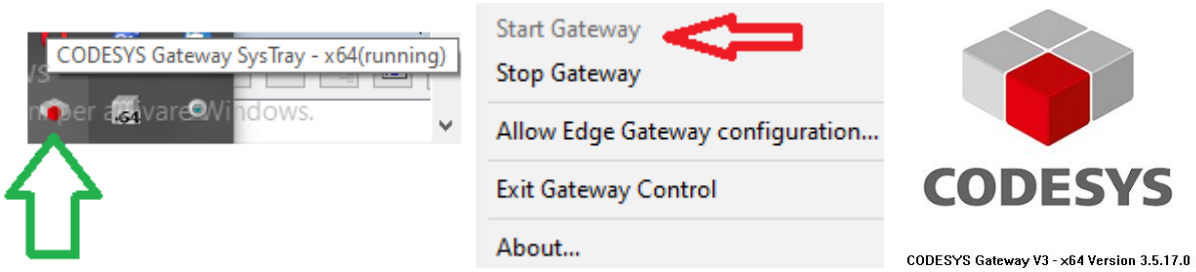
Installazione di RTE sul target P10L

Vediamo come scaricare su un terminale video HMI P10L "vergine" (sia CM3 che CM4, ovvero Raspberry PI Compute Module 3 e 4) il supporto real time, ovvero l'RTE (Real Time Engine) dal PC-host-Windows (x86, x64), dove sopra gira l'ambiente di sviluppo Codesys IDE, poi come scaricare l'applicazione di lavoro da noi sviluppata, e come viene gestita la visualizzazione grafica delle pagine dell'applicazione di lavoro.

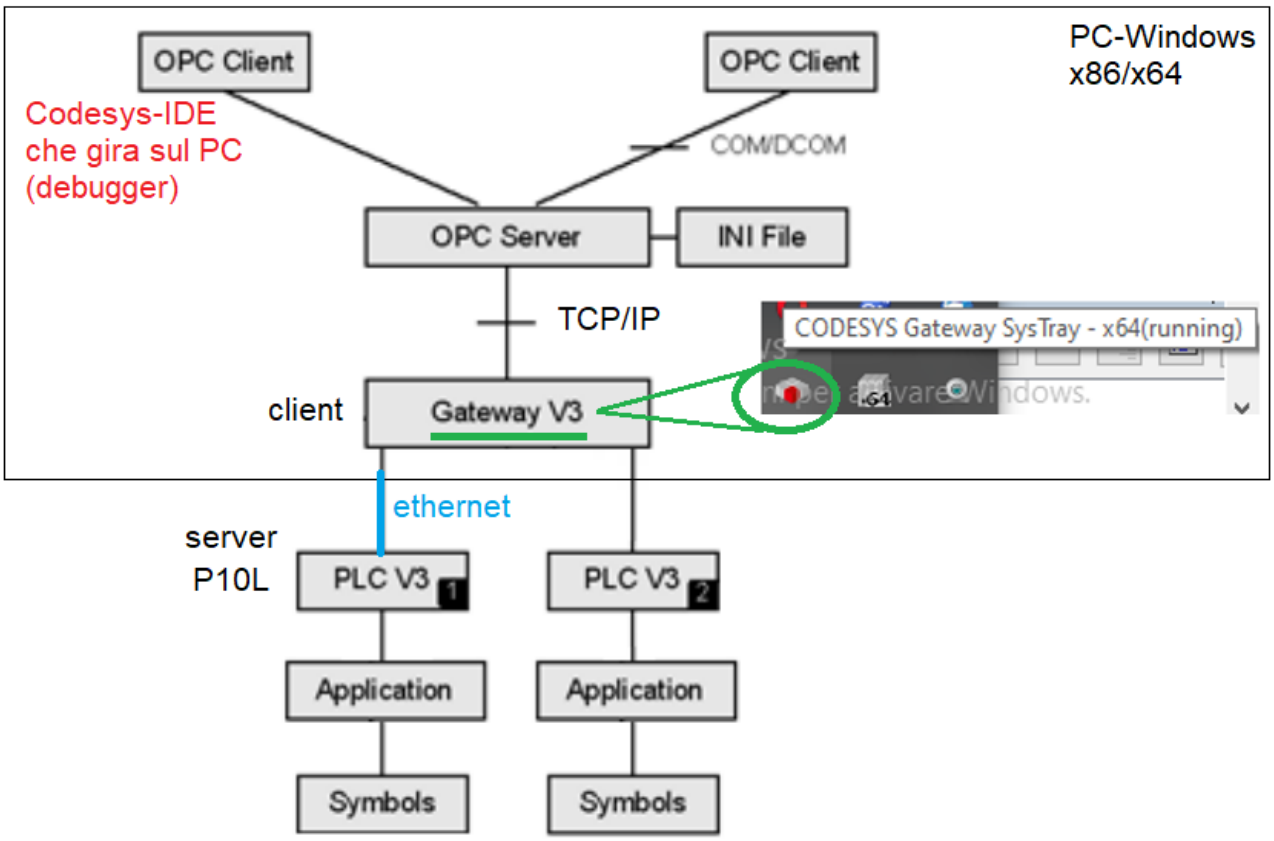
Per condurre queste prove, abbiamo collegato via ethernet il nostro PC-host-Windows, con sopra Codesys-IDE (versione 3.5, service pack 17, nel nostro caso), con la porta ethernet del P10L-CM4 (con P10L-CM3 sarebbe la stessa cosa).

Facciamo prima questo:



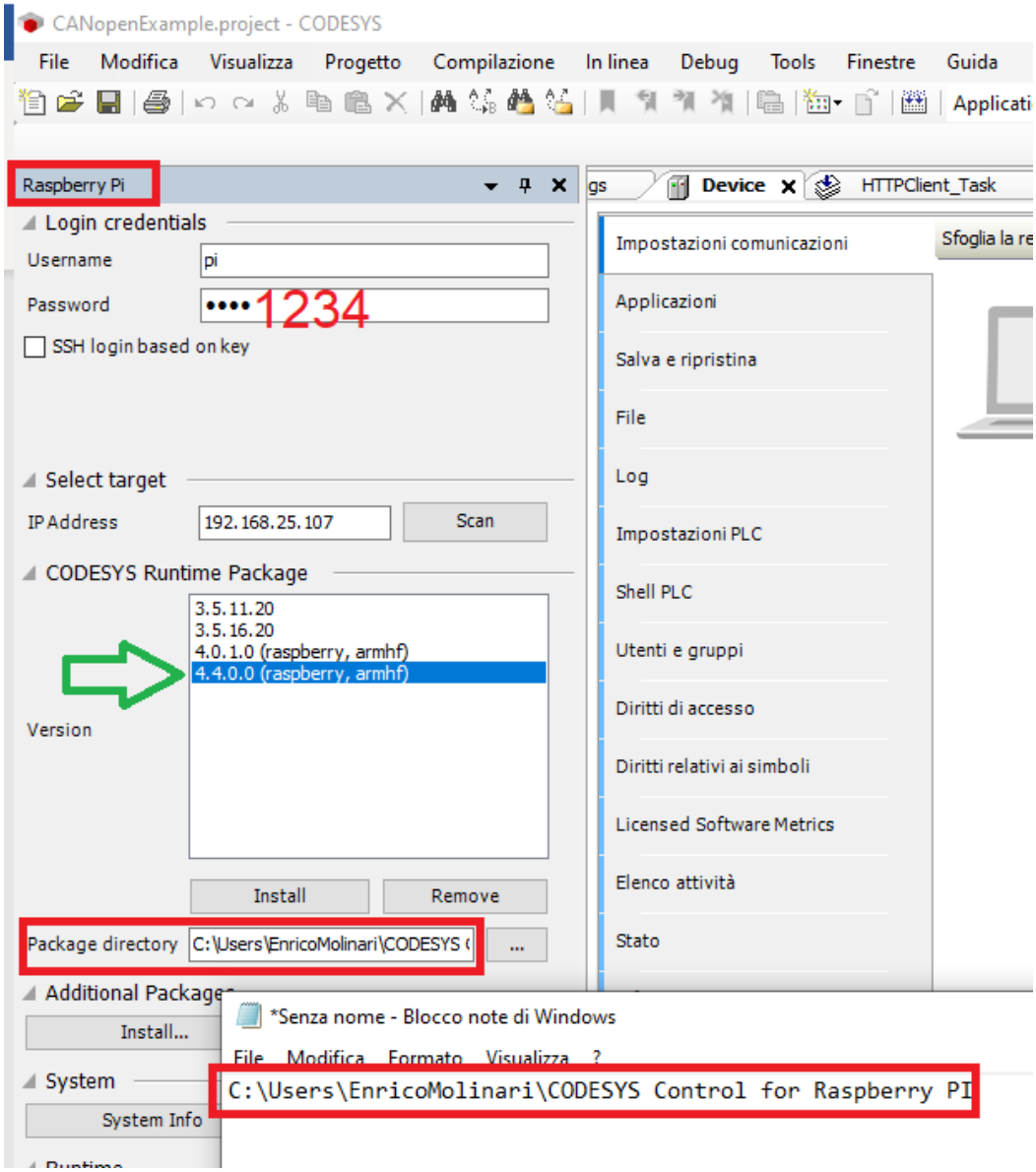


Uno schema funzionale di massima:



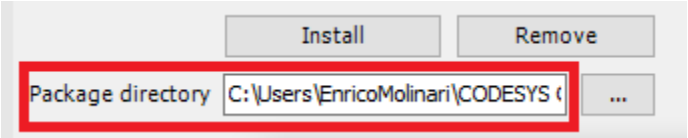
Da notare che il processo Codesys-Gateway-SysTray deve essere avviato (Start Gateway), se si vuole che il blocchetto Gateway abbia il pallino di color verde e non nero. Con Gateway avviato (pallino verde), Codesys-IDE può, via OPC-UA/TCP, comunicare con P10L (PLC).
All'inizio, ovvero con P10L vergine, è normale avere il blocchetto Raspberry PI con pallino nero, questo perché adesso, su P10L (= Raspberry PI by Syel), non abbiamo ancora installato il processo RTE.

Fare quanto segue:

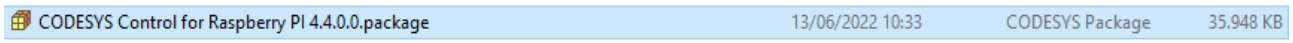


Stiamo quindi selezionando, nell'opportuno path del PC-host-Windows, il run-time RTE che vogliamo installare sul target P10L, in questo caso l'RTE più recente, ovvero il 4.4.0.0. L'IP è ovviamente quello del target P10L.

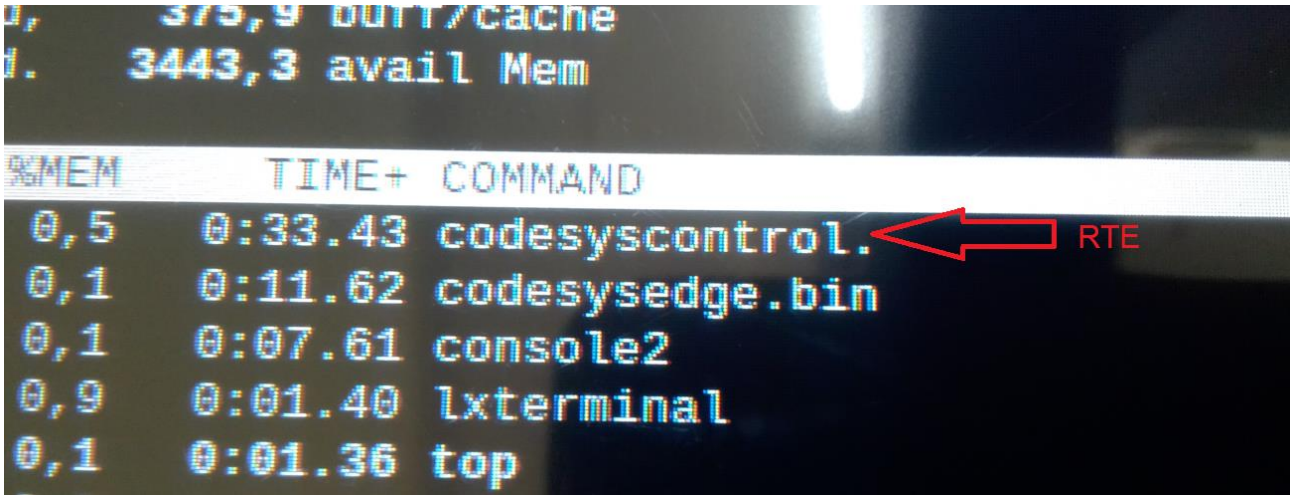
Clicchiamo sul bottone Install:



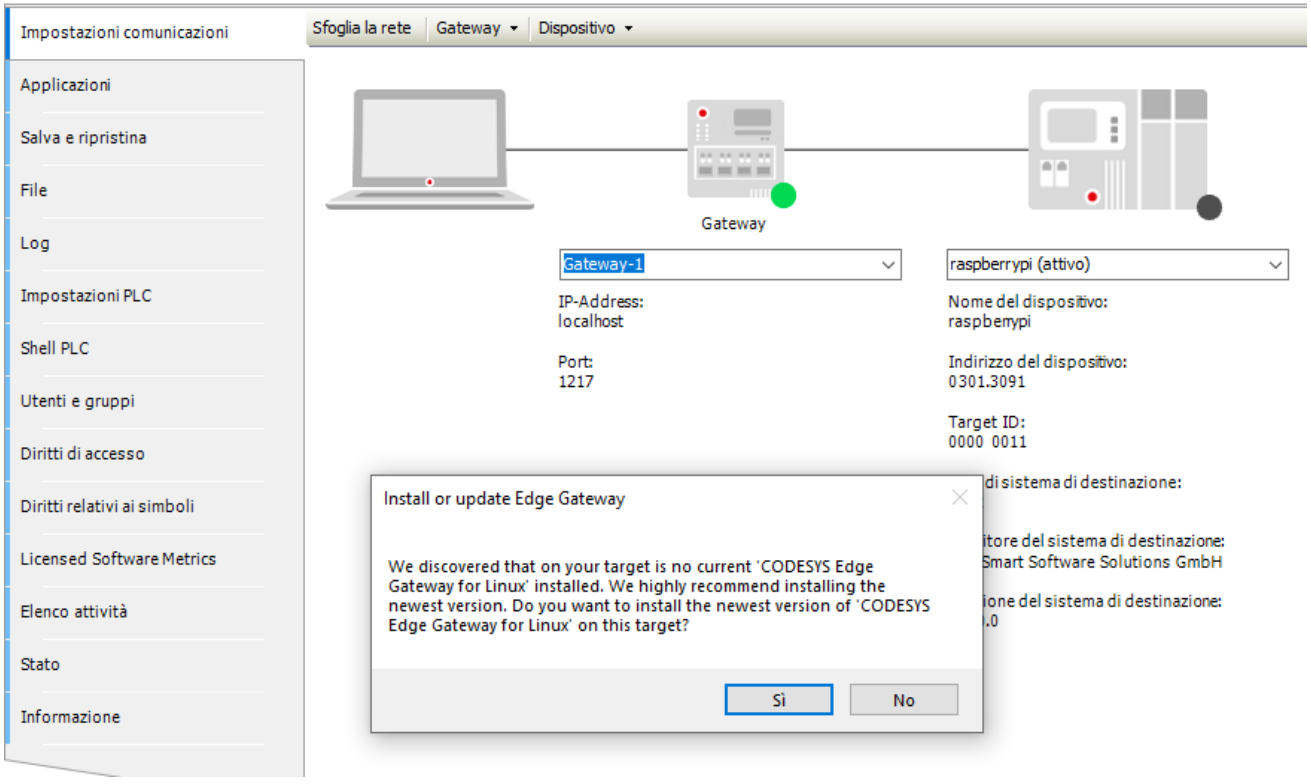
Alla fine del trasferimento, abbiamo installato sul target P10L questo pacchetto qui:



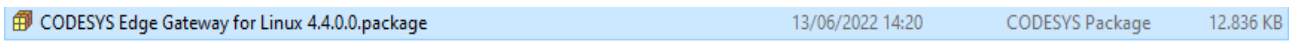
Sul target P10L, se battiamo il comando “top” da shell, dobbiamo vedere che il processo RTE è installato e correttamente in esecuzione, insieme a tutti gli altri processi concorrenti su Linux:



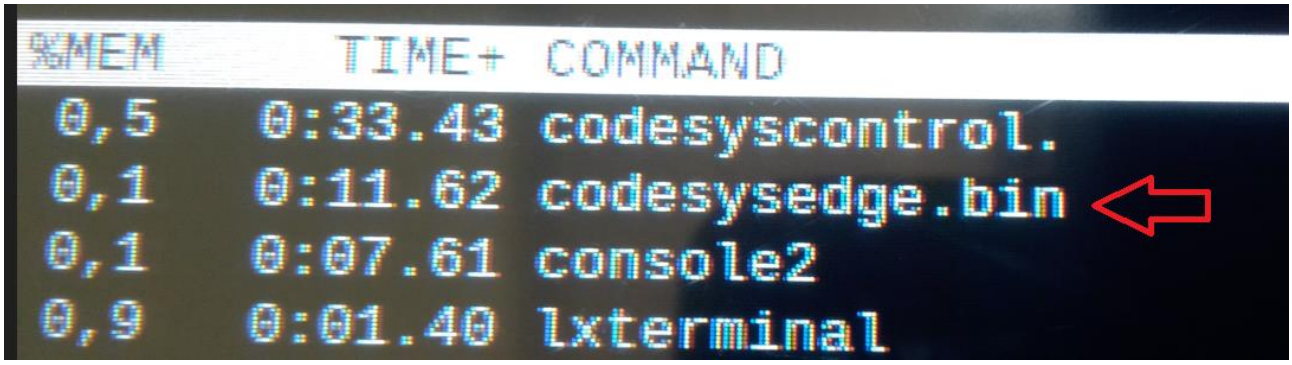
Adesso Codesys-IDE ci fa questa osservazione:



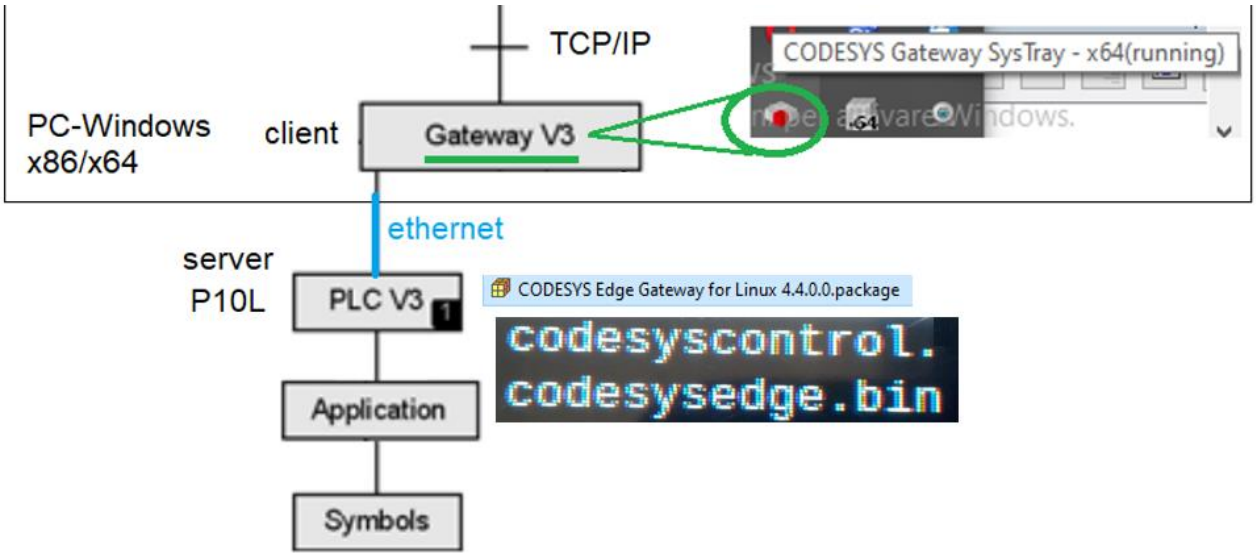
Noi clicchiamo SI, quindi si avvia l’installazione sul target P10L di questo secondo pacchetto:



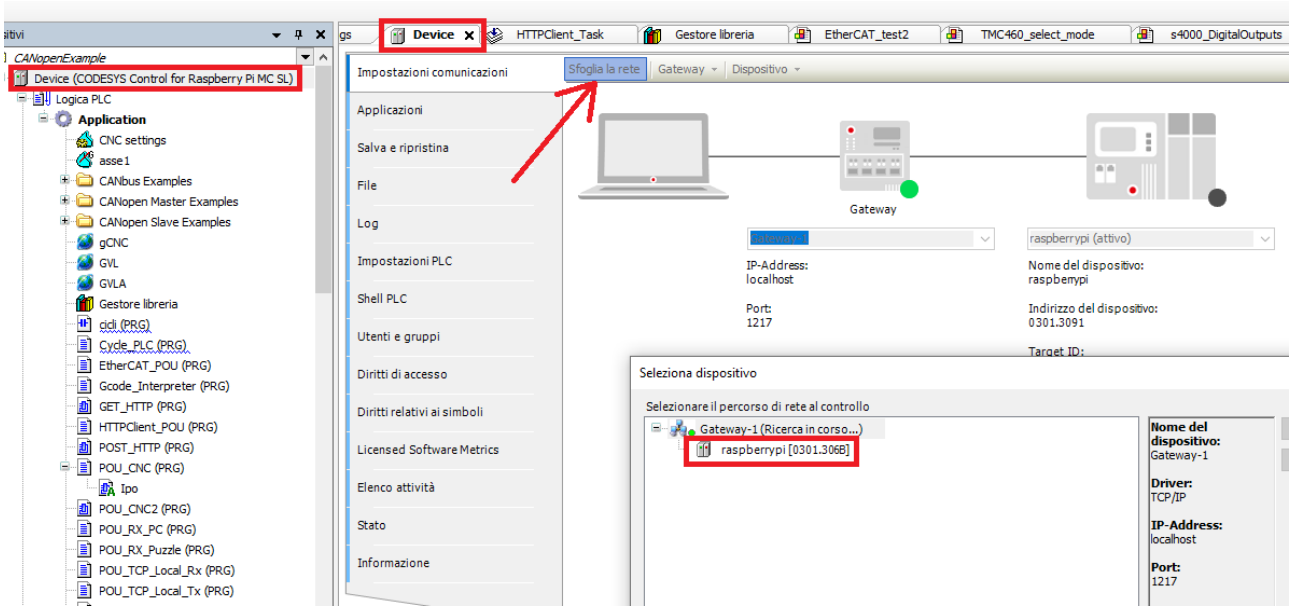
Sul target P10L, se battiamo il comando “top” da shell, dobbiamo vedere che il processo codesysedge.bin è installato e correttamente in esecuzione, insieme a tutti gli altri processi concorrenti su Linux:



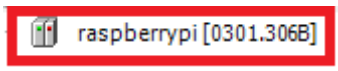
Questa è la situazione al momento:



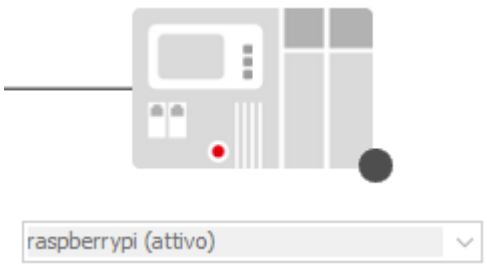
Adesso se diciamo al client OPC-UA/TCP di Codesys-IDE di sfogliare la rete, lui trova il server OPC-UA/TCP in ascolto sulla porta 1217, poiché il processo RTE appena scaricato sul target P10L è già operativo, ed è lui che implementa tale server TCP, che fra poco funzionerà prima come bootloader, e poi come supporto real-time.



Cliccare qui:

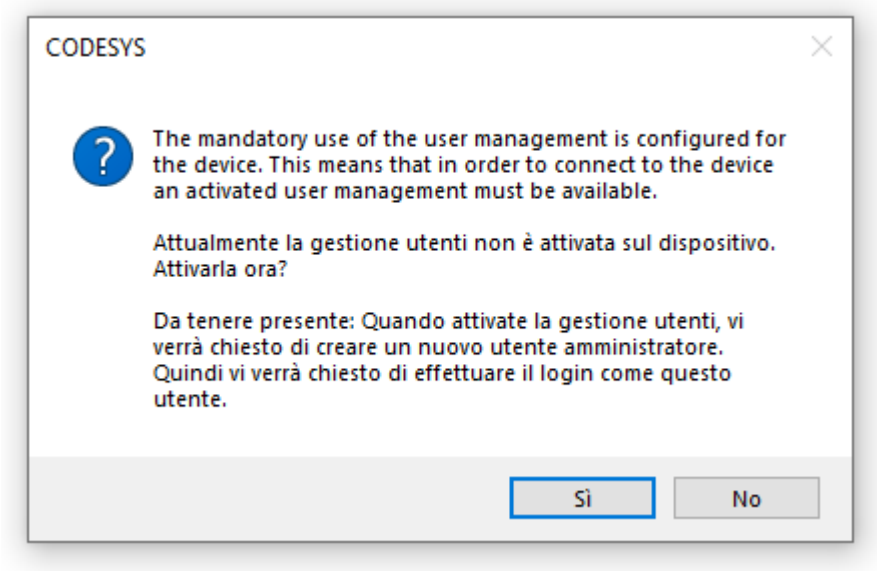


Per trasformare questo pallino nero:



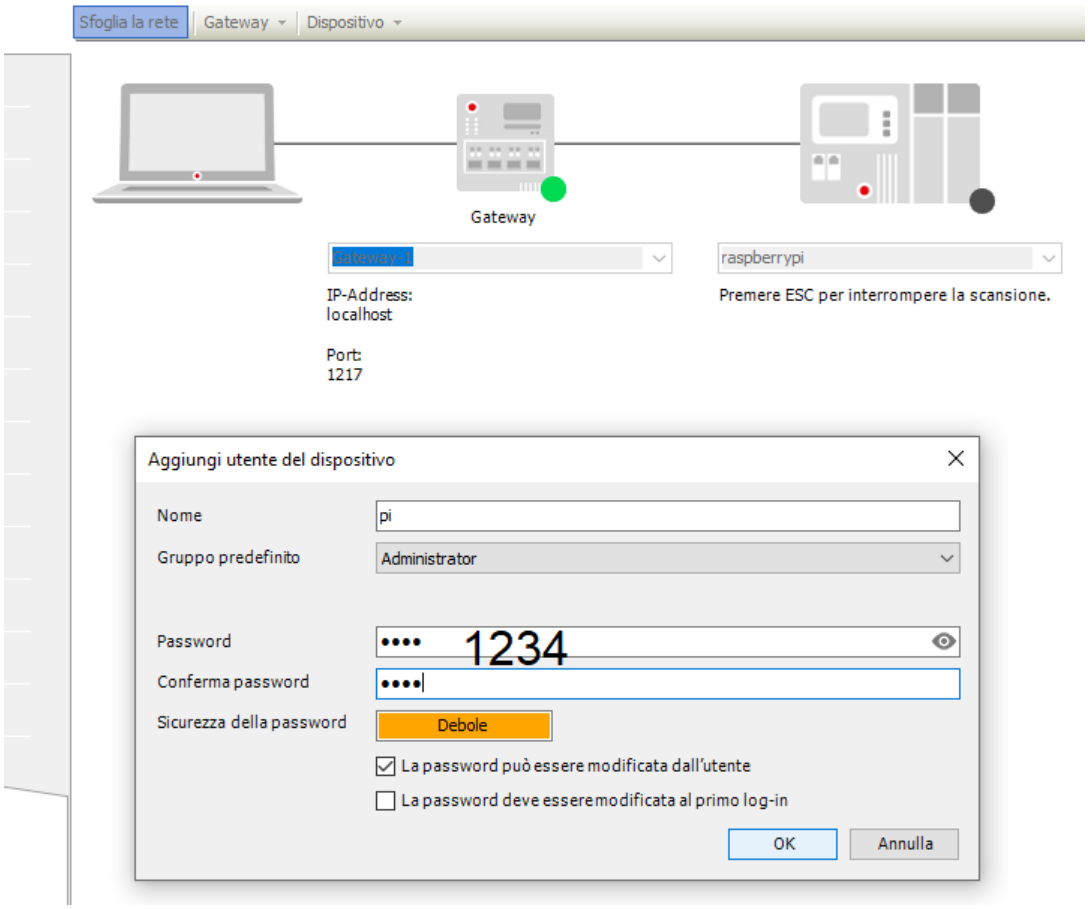
in verde.

Adesso Codesys ci dice questo:

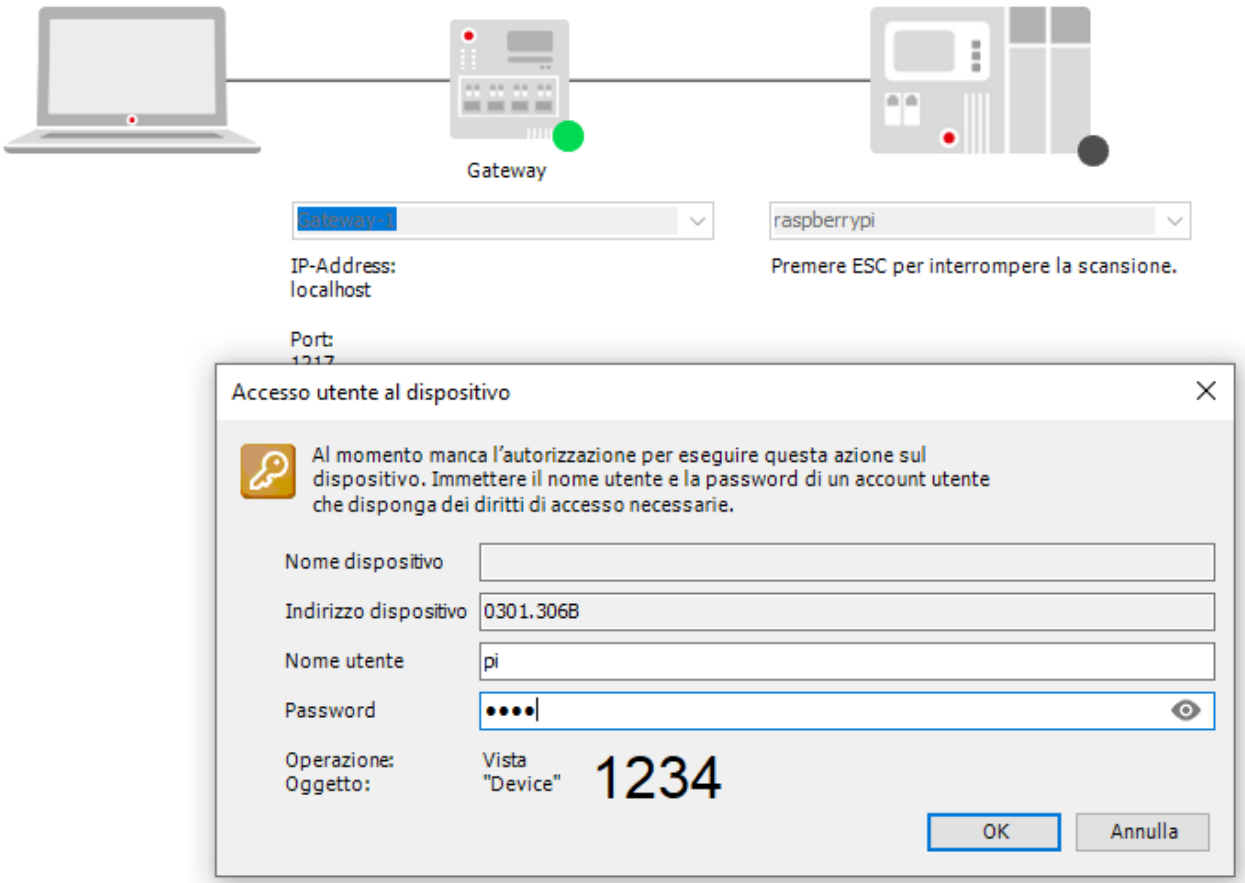


Clicchiamo su SI.

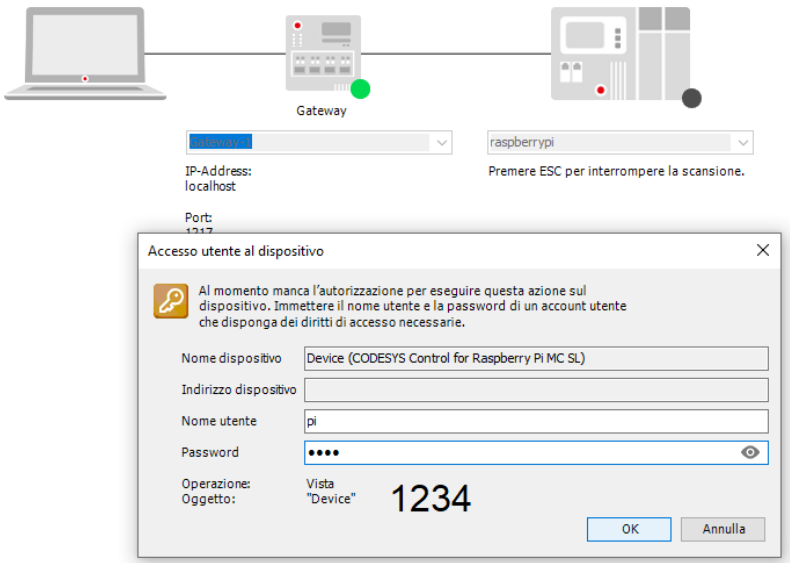
Poi facciamo così:



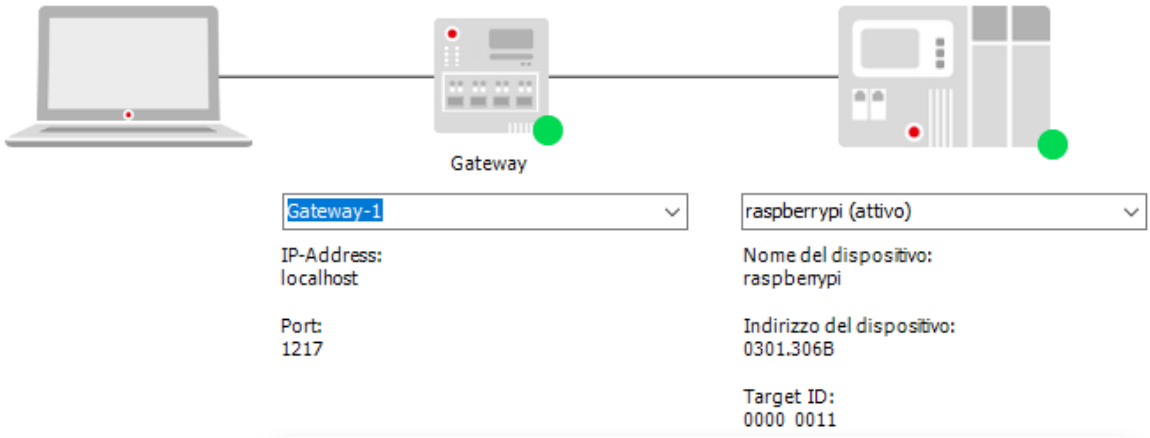
Riproviamo a collegarci con il target P10L, ma ci viene detto questo, e noi reinseriamo i dati, gli stessi dati appena inseriti:



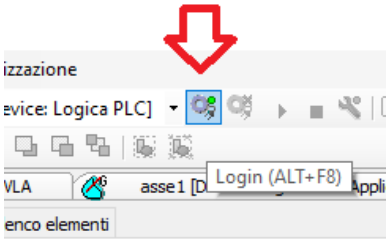
Adesso ci viene ripresentato un popup leggermente diverso, e noi inseriamo gli stessi dati:



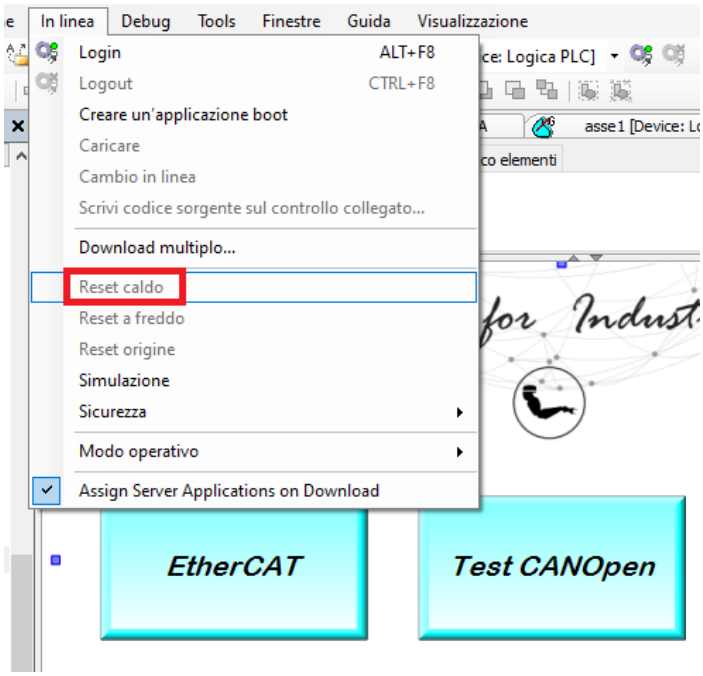
Adesso il target è pronto, nel senso che non solo il supporto a run-time RTE gira correttamente sul target P10L, ma il nostro PC-host-Windows, tramite il client OPC-UA/TCP di Codesys-IDE, è collegato correttamente al server OPC-UA/TCP implementato dall'RTE. PC e target si vedono, sono connessi, pallino verde.



Adesso possiamo scaricare l'applicazione di lavoro sul target, facendo il Login:



Poi facciamo il reset a caldo, per portare il program-counter alla prima istruzione dell'applicazione di lavoro:



E adesso clicchiamo su run application:

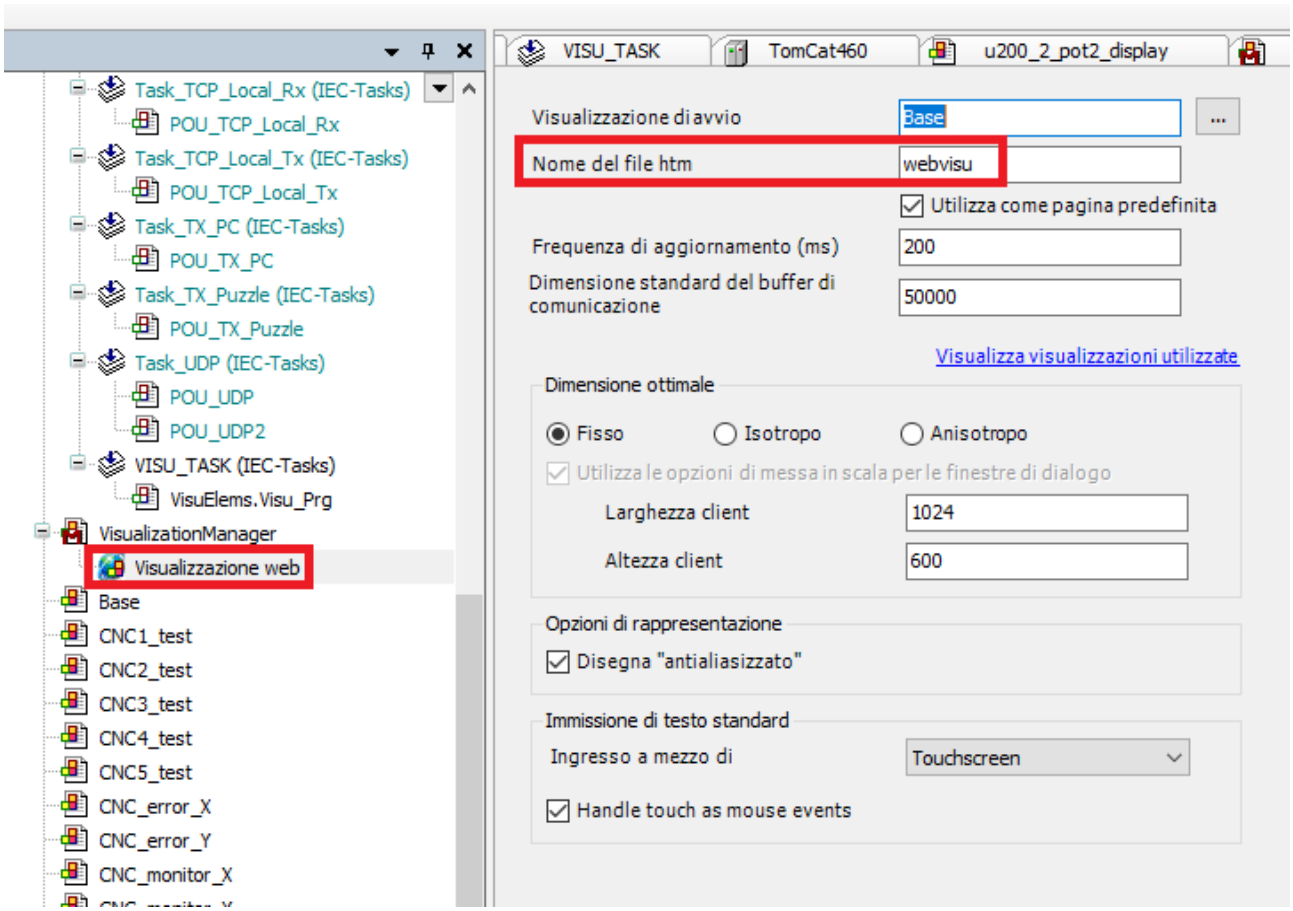
Adesso l'applicazione gira.

RTE è un framework, una engine, un motore sotterraneo che ha prima funzionato come bootloader TCP in ascolto su una porta, grazie al quale siamo riusciti ad inviare al target l'applicazione di lavoro, e da ora in poi RTE funziona appunto come framework, che fa il parsing di uno script, dove lo script è proprio l'applicazione di lavoro appena inviata.

RTE è un po' come .NET su PC-Windows, l'applicazione di lavoro è un po' come un programma scritto in C-Sharp.

Visualizzazione delle pagine grafiche sul target P10L

Guardiamo qui:



L'applicazione ad alto livello, da noi sviluppata, mette a disposizione un file "webvisu.htm".

Dobbiamo quindi far sì che Linux, su target P10L, lanci un qualunque internet-browser, ad esempio Chromium-Browser, e far sì che tale internet-browser apra (richieda), in local-host (127.0.0.1), la risorsa "webvisu.htm".

In sostanza:

su Chromium-Browser su P10L possiamo battere, sulla barra dell'URL:

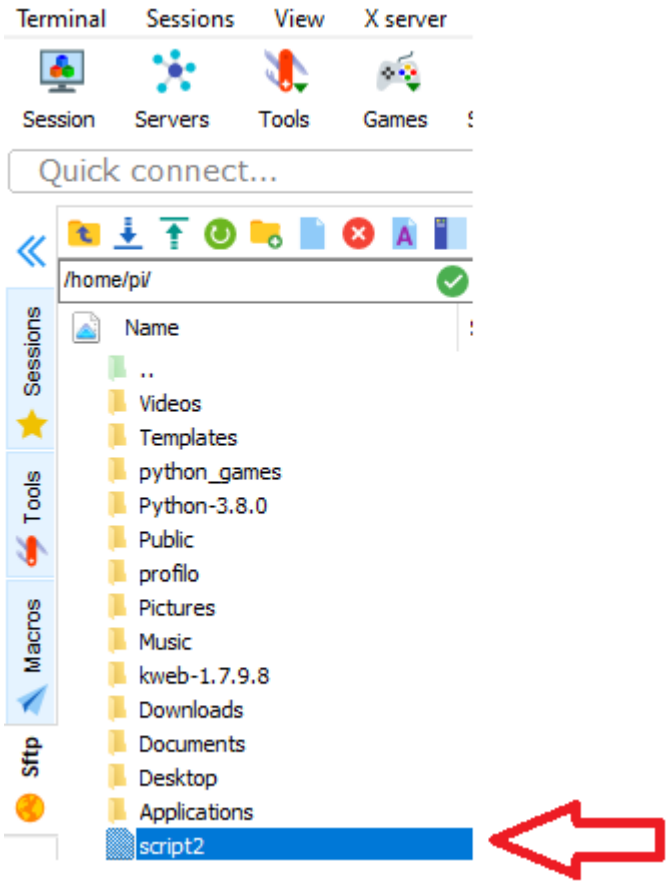
http://127.0.0.1:8080/webvisu.htm

/webvisu.htm = end-point, risorsa di cui il browser fa la GET:

GET /webvisu.htm HTTP/1.1
Host: 127.0.0.1:8080
From: Chromium-Browser

Durante lo sviluppo dell'applicazione di lavoro, si può pensare di procedere in questo modo:

mettiamo, in un qualunque path del filesystem di P10L, uno script:



Appena avviato il P10L, lanciamo una shell (di default, sia in "/home/pi", poiché noi siamo utente "pi"), dopodichè battiamo il comando:

sh script2

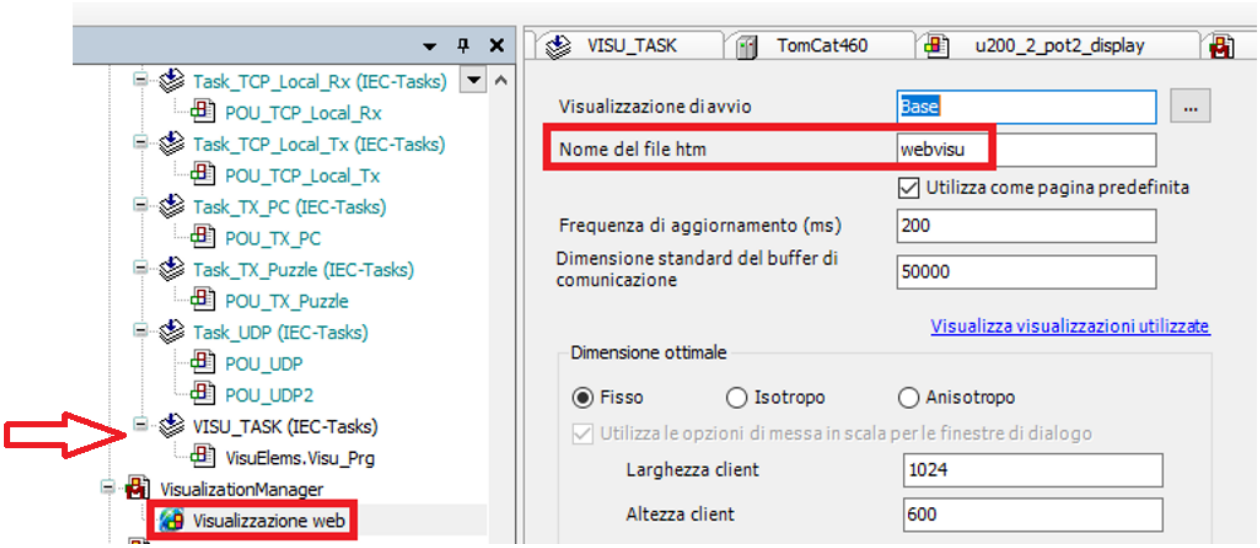
questo è il contenuto di script2:

```
chromium-browser --noerrdialogs --allow-running-insecure-content --window-position=0,0
--start-maximixed --window-size=1024,600 --kiosk --incognito http://localhost:8080/webvisu.htm
```

Pertanto viene lanciato l'internet-browser Chromium (il quale si apre e ricopre tutto l'LCD, 1024 x 600 pixels, in modalità kiosk, ovvero full-screen e foreground, ciò la finestra di Chromium viene avanti a tutto il resto, e qualunque altra pagina/finestra/popup grafici resta sotto), passandogli tutta una serie di parametri, e in particolare gli viene passato il parametro URL:

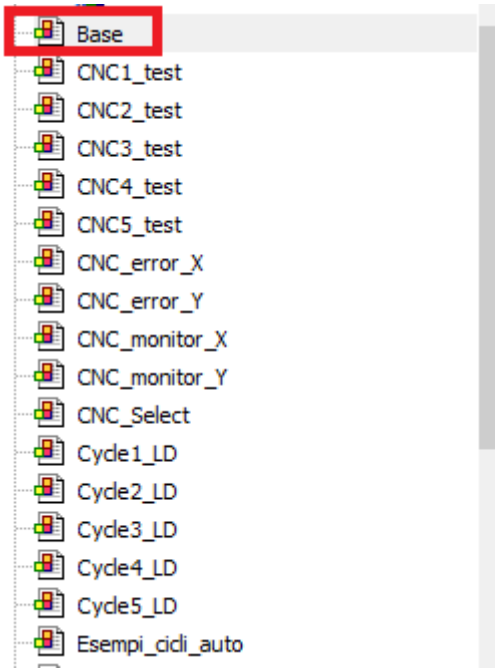
```
http://localhost:8080/webvisu.htm
```

Chromium è quindi un http-client che si connette con l'http-server "localhost" (lo stesso P10L), server che ascolta sulla porta 8080, server implementato dall'applicazione di lavoro da noi sviluppata, o meglio, dal task "VISU_TASK" (libreria "TargetVisu"):

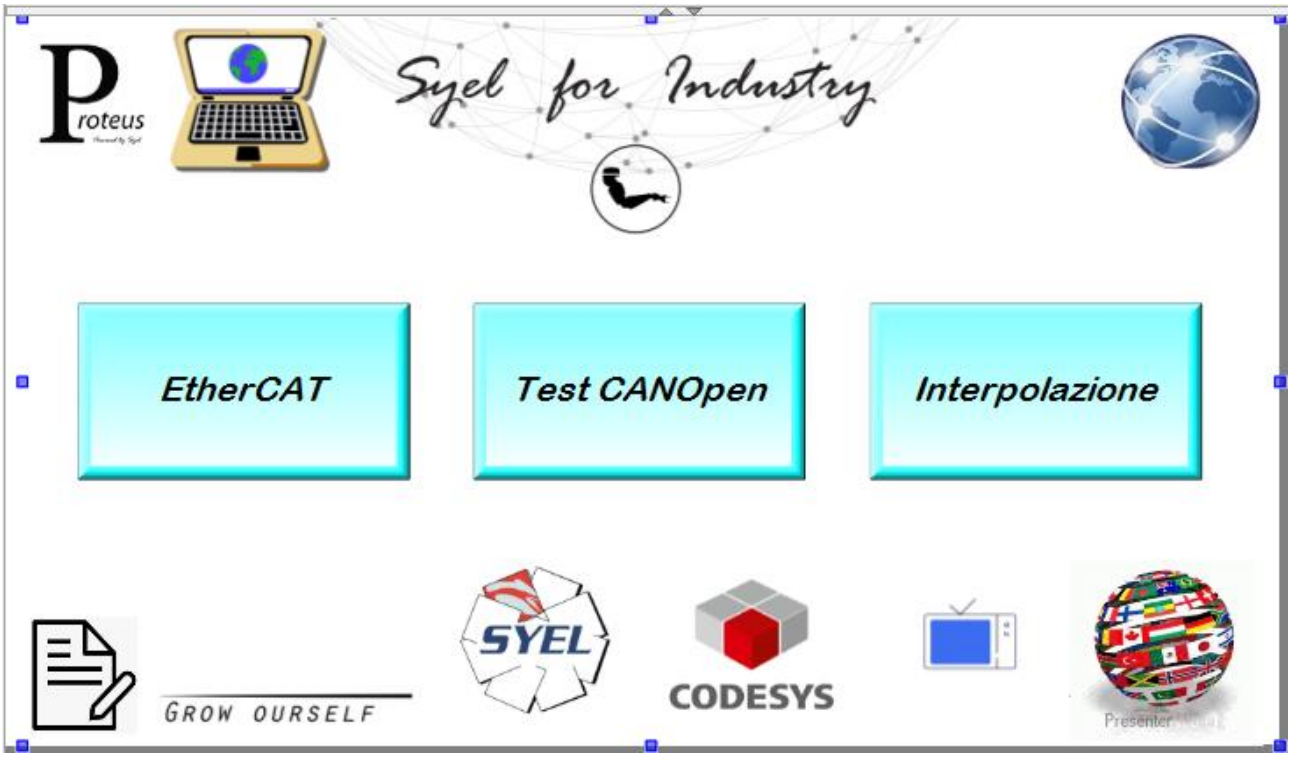


Server che restituisce, al client Chromium, la risorsa webvisu.htm richiesta.

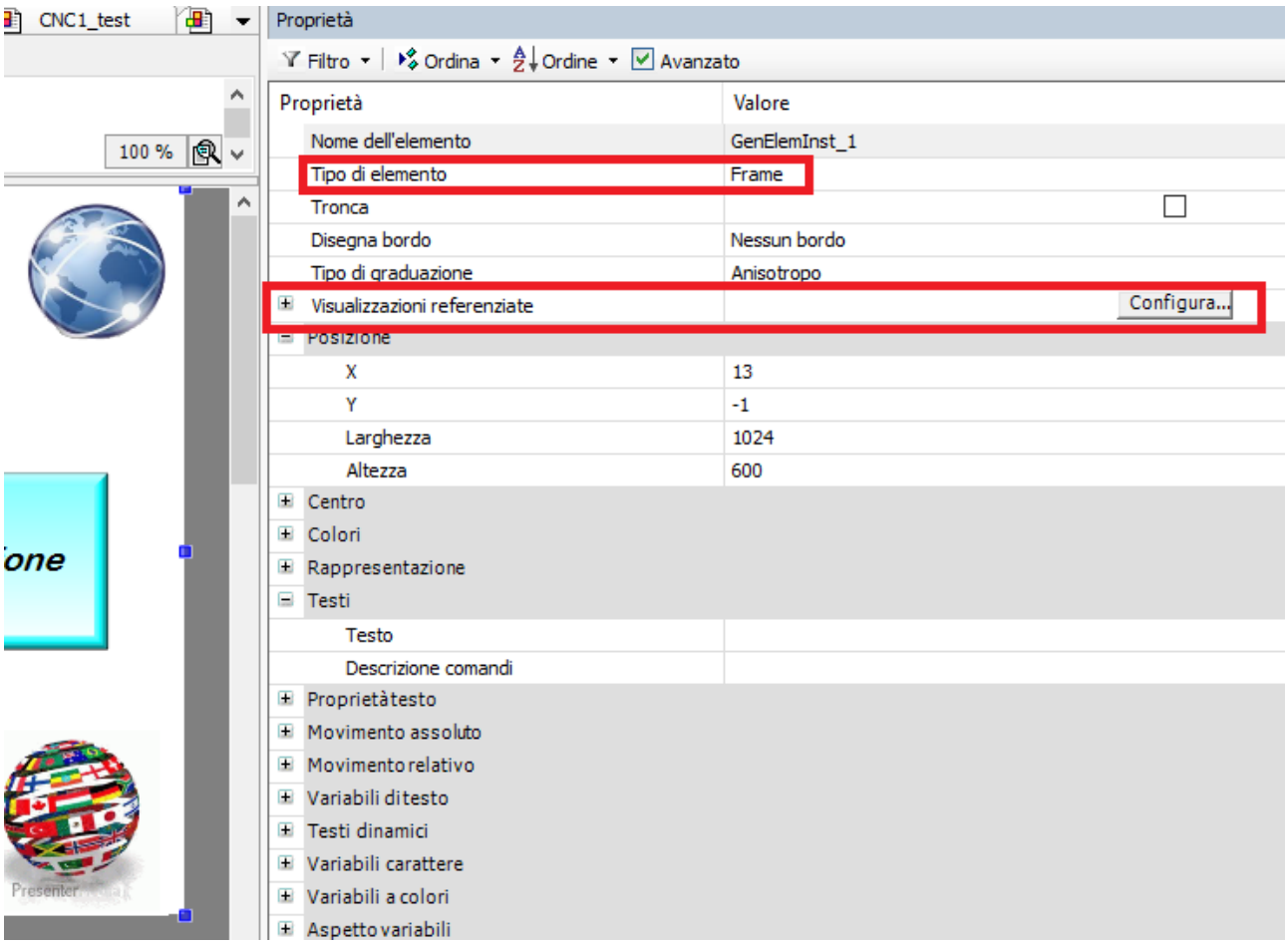
Il client, a questo punto, fa il parsing del file webvisu.htm, e da questo parsing viene fuori la visualizzazione grafica delle pagine dell'applicazione di lavoro:

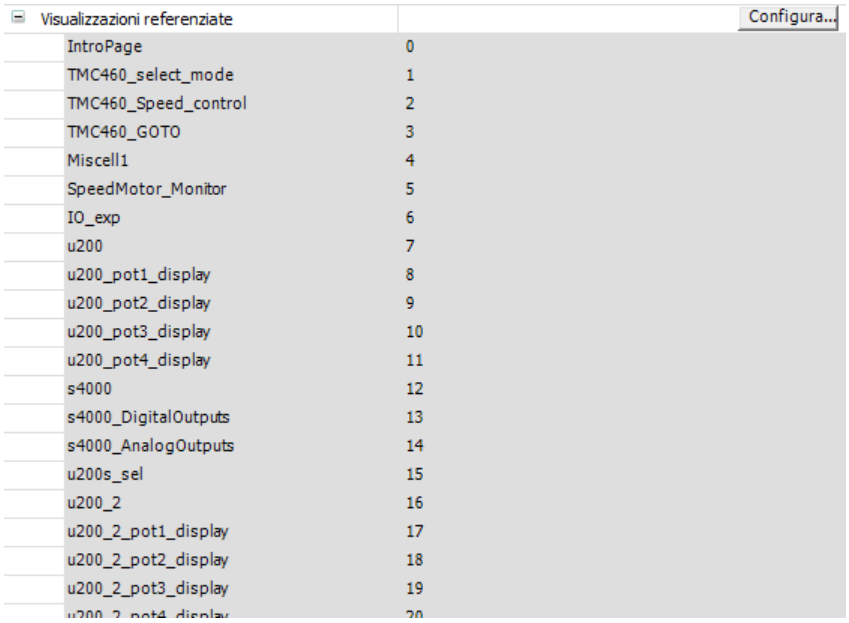


Il risultato è questo:



"Base" è la pagina di intro, in cui dobbiamo mettere un oggetto di tipo "Frame" delle stesse dimensioni 1024 x 600 dello schermo del P10L e delle pagine grafiche, grazie al quale possiamo indicizzare tutte le pagine del programma:





L’avvio manuale dello script “script2”, da shell, può andar bene in fase di nostro sviluppo, ma quando si consegna il P10L al cliente, questo appena accende il P10L, deve vedere l’avvio automatico di Chromium-Browser, quindi dobbiamo fare in modo che tale script venga interpretato da Linux (o meglio, dal programma “sh”), in modo automatico, all’avvio del P10L.

Ci sono svariati modi, la descrizione dei quali esula dallo scopo di questo documento: ad esempio uno di questi è di scrivere la riga dello script di cui sopra dentro il file “.profile” che si trova in “/home/pi”, poiché lo script “.profile” viene interpretato automaticamente appena l’utente pi effettua il login.