[**pycomm3**](https://pycomm3.readthedocs.io/en/latest/#module-pycomm3) **- Una libreria Python Ethernet/IP per la comunicazione con i PLC Allen-Bradley.**

Lo sviluppo di questa libreria ha lo scopo di fornire un accesso rapido e conveniente per la lettura / scrittura dei dati all'interno dei PLC Allen-Bradley.

Questa libreria è supportata su Python 3.6.1 e versioni più recenti e contiene 3 driver diversi

**LogixDriver**

Questo è il driver principale per questa libreria, supporta PLC ControlLogix, CompactLogix e Micro800.

**Driver SLC**

Questo driver può essere utilizzato per la lettura/scrittura di file di dati in PLC SLC 500 o MicroLogix. Alcune delle funzionalità più avanzate o automatiche non sono supportate.

**CIPDriver**

Questa è la classe base per gli altri due driver, gestisce alcuni servizi condivisi comuni. Può anche essere utilizzato per la messaggistica CIP generica ad altri dispositivi non PLC.

**Documentazione**

Questo documento illustra una panoramica di base della libreria, la documentazione completa è disponibile [cliccando qui](https://pycomm3.readthedocs.io/en/latest/).

**LogixDriver**

### Caratteristiche

* API semplice, solo 1 metodo di lettura e 1 metodo di scrittura per i tag.
  + non richiede l'utilizzo di metodi diversi per diversi tipi di dati
  + richiede solo il nome del tag, non sono richieste altre informazioni all'utente
  + gestisce automaticamente le dimensioni di richiesta/risposta per compattare il numero di richieste in un unico pacchetto
  + gestisce automaticamente le richieste frammentate di tag di grandi dimensioni che non possono essere adattate a un singolo pacchetto
  + entrambi supportano la lettura/scrittura completa della struttura (UDT, AOI, ecc.))
    - Per la lettura, Tag.value sarà un dict di {attribute: value}
    - Per scrivere il valore dovrebbe essere una sequenza (elenco, tupla) di valori per ogni attributo, annidato in base alle esigenze
      * non scrive in parte, il valore deve corrispondere alla struttura completa
      * non consigliato per il tipo incorporato (TIMER, CONTROL, COUNTER, ecc.))
* generic\_message funzionalità aggiuntive non implementate direttamente
  + funziona in modo simile all'istruzione MSG in Logix, argomenti simili alle proprietà MESSAGE
  + testato per ottenere / impostare i parametri dell'azionamento
  + utilizzato internamente per implementare alcuni degli altri metodi (getget/set\_plc\_time, forward open/close, ecc.)
* tipi di dati semplificati
  + le stringhe utilizzano normali oggetti str Python, non richiedono la lettura/scrittura degli attributi LEN e DATA
  + Le matrici BOOL utilizzano normali oggetti bool Python, non richiedono un complicato spostamento di bit del valore DWORD
* caricare l'elenco dei tag e le definizioni dei tipi di dati dal PLC
  + nessun requisito per l'utente per determinare i tag disponibili (ad esempio da un'esportazione L5X)
  + i tag dell'ambito del controllore per impostazione predefinita, i tag dell'ambito del programma sono facoltativi
  + le definizioni sono necessarie per i metodi di lettura/scrittura
* abilita/disabilita/automaticamente diverse funzionalità in base al PLC di destinazione
  + Extended Forward Open (EN2T o più recente e v20 +)
  + Indirizzamento delle istanze dei simboli (Logix v21 +)
  + rilevamento di Micro800 e disabilita le funzionalità non supportate (percorso CIP, es. inoltro aperto, indirizzamento istanza, ecc.)

### Utilizzo di base

Connettersi a un PLC e ottenere alcune informazioni di base su di esso. L'argomento path è l'unico richiesto e ha 3 forme:

* Solo indirizzo IP (10.20.30.100) - Utilizzare se il PLC si trova nello slot 0 o se ci si connette a CompactLogix
* Indirizzo IP/Slot (10.20.30.100/1) - Utilizzare se il PLC non è nello slot 0
* Percorso routing CIP (10.20.30.100/backplane/3/enet/10.20.40.100/backplane/0) – Utilizzo per routing più complessi
  + i primi 2 esempi verranno sostituiti automaticamente con il percorso completo
  + enet/backplane (o bp) sono per la selezione delle porte

### Esempio

from pycomm3 import LogixDriver

with LogixDriver('10.20.30.100/1') as plc:

print(plc)

# OUTPUT:

# Program Name: PLCA, Device: 1756-L83E/B, Revision: 28.13

print(plc.info)

# OUTPUT:

# {'vendor': 'Rockwell Automation/Allen-Bradley', 'product\_type': 'Programmable

Logic Controller', 'product\_code': 166, 'version\_major': 28, 'version\_minor':

13, 'revision': '28.13', 'serial': 'FFFFFFFF','device\_type': '1756-L83E/B',

'keyswitch': 'REMOTE RUN', 'name': 'PLCA'}

Di default, quando si crea l'oggetto LogixDriver, esso aprirà una connessione al plc, leggerà il nome del programma, otterrà le informazioni del controllore e tutti i tag dell'ambito del controllore. Leggendo prima l'elenco dei tag, questo ci consente di memorizzare nella cache tutte le informazioni sul tipo/struttura di tag, inclusi gli ID di istanza per tutti i tag. Queste informazioni consentono ai metodi di lettura/scrittura di richiedere solo il nome del tag. Se il tuo progetto richiederà tag nell'ambito del programma, assicurati di impostare init\_program\_tags kwarg. Per impostazione predefinita, verranno recuperati e memorizzati nella cache solo i tag con ambito controller.

**Tag di lettura/scrittura**

Leggere o scrivere tag è semplice come chiamare i metodi di lettura e scrittura. Entrambi i metodi accettano un numero qualsiasi di tag e impacchetteranno automaticamente più tag in un Multiple Service Packet Service (0x0A) assicurandosi di rimanere al di sotto della dimensione della connessione. Se è presente un valore di tag che non può rientrare nel pacchetto di richiesta / risposta, gestirà automaticamente quel tag in modo indipendente utilizzando le richieste Read Tag Fragmented (0x52) o Write Tag Fragmented (0x53).

Entrambi i metodi restituiranno oggetti Tag per il successo o il fallimento dell'operazione.

### Esempio

with LogixDriver('10.20.30.100') as plc:

plc.read('tag1', 'tag2', 'tag3') # legge più tag

plc.read('array{10}') # legge 10 elementi di un array dalla posizione 0

plc.read('array[5]{20}) # legge 20 elementi di un array dalla posizione 5

plc.read('string\_tag') # legge un tag stringa

# le scritture richiedono una sequenza di tuple di [(tag name, value), ... ]

plc.write(('tag1', 0), ('tag2', 1), ('tag3', 2)) # scrive più tag

plc.write(('array{5}', [1, 2, 3, 4, 5])) # scrive 5 elementi di un array dalla

posizione 0

plc.write(('array[10]{5}', [1, 2, 3, 4, 5])) # scrive 5 elementi di un array dalla

posizione 10

plc.write(('string\_tag', 'Hello World!')) # scrive una stringa

plc.write(('string\_array[2]{5}', 'Write an array of strings'.split())) # scrive un

array di stringhe dalla posizione 2

Nota:

I nomi dei tag sia per la lettura che per la scrittura fanno distinzione tra maiuscole e minuscole e devono essere gli stessi di quelli denominati nel controller.

**Tag di lettura/scrittura**

Le definizioni dei tag vengono caricate automaticamente dal controller durante la connessione. Ciò consente ai metodi di lettura/scrittura di funzionare. Queste definizioni contengono informazioni come ID istanza e dimensioni e composizione della struttura. Le definizioni dei tag sono accessibili dall'attributo tags. La proprietà tags è una dict di {nome tag: definizione}.

Le definizioni per le strutture sono accessibili dall'attributo data\_types. Questi includono cose come tipi di dati definiti dall'utente (UDT), istruzioni add-on (AOI), stringhe e tipi predefiniti (TIMER, COUNTER, ecc.). Per i tag di struttura (tag ['tag\_type'] == 'struct'), la definizione del tipo di dati verrà memorizzata nell'attributo data\_type. (i tag "atomic" avranno solo una stringa con il nome del tipo di dati: "DINT", "REAL", ...).