CommunicationAPIS7

Net and Light

2023 - 2024

Résumé

Cette documentation décrit le programme informatique intitulé "CommunicationAPIS7.py". Il comprend une vue d'ensemble, une description des fonctionnalités, une architecture, et des instructions d'installation et d'utilisation.

Table des matières

1 Introduction

Cette documentation présente le script "CommunicationAPIS7.py", qui utilise la bibliothèque Snap7 pour gérer la communication avec des automates programmables logiques (PLC) Siemens. Le script permet de lire et d'écrire des données mémoire sur les PLC.

2 Vue d'ensemble du Projet

Le projet "CommunicationAPIS7.py" permet de se connecter à des PLC Siemens, de lire et d'écrire des données mémoire à l'aide de la bibliothèque Snap7. Il utilise également des expressions régulières pour extraire des adresses logiques et inclut des fonctions utilitaires pour gérer différentes opérations de données.

3 Architecture du Système

3.1 Description Globale

L'application est composée des composants suivants :
— **Snap7** : Pour la communication avec les PLC Siemens.
— **re** : Pour les opérations sur les chaînes de caractères utilisant des expressions régulières.
— **variablesAPI** : Pour obtenir les informations de configuration des variables.

3.2 Composants

Le script est structuré autour de plusieurs fonctions clés permettant de lire et d'écrire des données sur les PLC, en plus de gérer la connexion aux PLC.

4 Fonctionnalités

4.1 Fonctionnalité 1 : Définition des constantes

Le script définit des constantes pour les longueurs de différents types de données utilisés dans les opérations sur les PLC.

```
1 BYTE_LENGTH = 1  # Length of a Byte
2 WORD_LENGTH = 2  # Length of a Word (2 Bytes) for Word, Int
3 DOUBLE_WORD_LENGTH = 4  # Length of a Double Word (4 Bytes)
    for DWord, Real, IEC Time
4 BYTE_START_OFFSET = 0  # Starting byte offset for bit
    extraction
5 NO_DB = 0  # No specific DB (used for certain read/write
    areas)
```

Listing 1 – Définition des constantes

4.2 Fonctionnalité 2 : Fonction pour obtenir la longueur des types de données

La fonction `getLength` détermine et retourne la longueur du type de données du PLC.

```
def getLength(type):
    match type:
        case "Bool":
            return BYTE_LENGTH
        case "Int":
            return WORD_LENGTH
        case "Word":
            return WORD_LENGTH
        case "Byte":
            return DOUBLE_WORD_LENGTH
        case "Time":
            return DOUBLE_WORD_LENGTH
        case _:
            print("Invalid type")  # Handle unexpected type
               input
```

Listing 2 – Obtenir la longueur des types de données

4.3 Fonctionnalité 3 : Extraction des numéros d'une adresse logique

La fonction `extractNumbers` extrait les valeurs numériques d'une chaîne d'adresse logique à l'aide des expressions régulières.

```
def extractNumbers(logical_address):
    numbers = re.findall(r'\d+', logical_address)
    return [int(number) for number in numbers] if numbers
       else None  # Convert found numbers to integers and
       return the list, or return None
```

Listing 3 – Extraction des numéros d'une adresse logique

4.4 Fonctionnalité 4 : Lecture de la mémoire du PLC

La fonction `readMemory` lit la valeur d'une variable dans un PLC.

```
def readMemory(plc, variable):
    type = variablesAPI.variables[variable]["Data Type"]  #
       Get the data type of the variable from the API
    length = getLength(type)  # Get the length of the data
       based on its type
    logical_address = variablesAPI.variables[variable]["
       Logical Address"]  # Get the logical address of the
       variable
```

```
    address = extractNumbers(logical_address)  # Extract numerical address
    start_address = address[0] if address != None else None  # Determine the start address
    bit_offset = address[1] if len(address) == 2 else 0  # Determine the bit offset if available

    reading = plc.read_area(snap7.types.Areas.MK, NO_DB, start_address, length)  # Read data from PLC
    match type:
        case "Bool":
            value = snap7.util.get_bool(reading, BYTE_START_OFFSET, bit_offset)  # Get boolean value from the data
            return value
        case "Int":
            value = snap7.util.get_int(reading, BYTE_START_OFFSET)  # Get integer value from the data
            return value
        case "Byte":
            return reading  # Return raw byte data
        case "Time":
            value = snap7.util.get_time(reading, BYTE_START_OFFSET)  # Use snap7 utility to parse time
            return value
        case _:
            print("Invalid type")
            return None
```

Listing 4 – Lecture de la mémoire du PLC

4.5 Fonctionnalité 5 : Écriture dans la mémoire du PLC

La fonction `writeMemory` écrit dans la mémoire d'un PLC, la valeur voulue pour une variable.

```
def writeMemory(plc, variable, value):
    type = variablesAPI.variables[variable]["Data Type"]  # Get the data type of the variable from the API
    length = getLength(type)  # Get the length of the data based on its type
    logical_address = variablesAPI.variables[variable]["Logical Address"]  # Get the logical address of the variable
    address = extractNumbers(logical_address)  # Extract numerical address
```

```
6       start_address = address[0] if address != None else None  # Determine the start address
7       bit_offset = address[1] if len(address) == 2 else 0  # Determine the bit offset if available
8
9       reading = plc.read_area(snap7.types.Areas.MK, NO_DB, start_address, length)
10      match type:
11          case "Bool":
12              snap7.util.set_bool(reading, BYTE_START_OFFSET, bit_offset, value)  # Set boolean value in the data
13              plc.write_area(snap7.types.Areas.MK, NO_DB, start_address, reading)  # Write back the modified data
14          case "Byte":
15              plc.write_area(snap7.types.Areas.MK, NO_DB, start_address, value)  # Write byte data directly
16          case "Int":
17              value = value.to_bytes(length, byteorder='big')  # Convert integer to bytes
18              plc.write_area(snap7.types.Areas.MK, NO_DB, start_address, value)  # Write integer data
19          case "Time":
20              snap7.util.set_time(reading, BYTE_START_OFFSET, value)  # Set time data
21          case _:
22              print("Invalid type")
```

Listing 5 – Écriture dans la mémoire du PLC

5 Instructions d'Utilisation

Les parties de code en commentaires permettent de tester les fonctionnalités du programme en dehors du projet global.

```
# Exemple de connexion à un PLC et de lecture de la mémoire
plc_ip_addresses = ['172.16.2.80']  # Liste des adresses IP des PLC
plcs = []  # Liste pour stocker les objets client Snap7

for ip_address in range(len(plc_ip_addresses)):
    plcs.append(snap7.client.Client())  # Création d'un nouveau client Snap7
    plcs[ip_address].connect(plc_ip_addresses[ip_address], 0, 1)  # Connexion au PLC

# Lecture d'une variable depuis un PLC connecté
for plc in plcs:
    print(readMemory(plc, "Mw_API_CVEntree"))  # Remplacer "Mw_API_CVEntree" par le nom de la variable souhaitée
```

Listing 6 – Exemple d'utilisation

6 Conclusion

Ce script fournit une solution simple pour interagir avec les PLC Siemens en utilisant Snap7. Il permet de lire et d'écrire des données mémoire de manière flexible et peut être adapté pour des besoins spécifiques de surveillance et de contrôle.

Références

[1] Snap7 Documentation, `http://snap7.sourceforge.net/`.
Python-Snap7, `https://python-snap7.readthedocs.io/en/latest/index.html`.

[2] Python re Module Documentation, `https://docs.python.org/3/library/re.html`.

[3] Variables API Documentation, source interne.