Table of contents

Python-Laufzeitumgebung

Python-Laufzeitumgebung

Python Runtime App für ctrlX CORE – Grundlagen

Die Python Runtime App enthält einen Python-Interpreter, der speziell dafür erweitert wurde, hocheffizient mit der ctrlX MOTION zu interagieren. Damit können Skripte abgearbeitet werden, die Bewegungskommandos vorgeben und auf den Zustand der ctrlX MOTION reagieren.

Skript-Steuerung über Skript-Manager

In der ctrlX CORE ist ein generischer Skript-Manager integriert. Dieser kann Skript-Interpreterinstanzen anlegen und steuern. Eine Beschreibung des Skript-Managers ist in der Dokumentation ctrlX CORE Runtime Anwendungsbeschreibung Kapitel Script Parser/Interpreter zu finden.

Um ein Python-Skript in der Python-Laufzeitumgebung auszuführen, müssen diese Schritte durchgeführt werden:

1. Skript in die Solution kopieren (siehe Abschnitt "Suchpfade für Python-Module")
2. Python-Interpreterinstanz per Skript-Manager anlegen
3. Skript in dieser Instanz ausführen lassen (siehe Abschnitt "Skriptausführung")

   (Die Python-Interpreterinstanz kann immer wieder benutzt werden und muss nur einmalig angelegt werden).

Integrierte Bibliotheken

In die Python-Laufzeitumgebung sind zwei Bibliotheken integriert:

- motion – Vorgabe von Kommandos und Auslesen von Zuständen der ctrlX MOTION, siehe ↘ „Integrierte Bibliothek – motion“
- datalayer – Einfacher Zugriff auf den Data Layer, siehe ↘ „Integrierte Bibliothek – datalayer“
  Zusätzlich sind alle Python-Bibliotheken vorhanden, die standardmäßig in Python integriert sind (z. B. sys).

Fehlerbehandlung

Die Fehlerbehandlung der beiden Bibliotheken erfolgt über Python-Exceptions. Wann immer ein Funktionsaufruf fehlschlägt, wird eine Exception ausgelöst (typischerweise ein RuntimeError). Sie können diese Exception erfassen und Ihre eigene Fehlerbehandlung vornehmen. Wenn die Exception nicht erfasst wird, bricht das Skript ab und alle angehängten (siehe motion.attach_obj()) Kinematiken und Achsen werden gestoppt (mit einem leichten Fehler).

Namensargumente

Die Namensargumente in den Funktionen beider Bibliotheken können außer der Reihe verwendet werden. Alle Namensargumente können als unbenannte Argumente verwendet werden (dann ist die Reihenfolge relevant).

Zusätzliche Python-Bibliotheken

In der Python-Laufzeitumgebung können nur Python-Bibliotheken genutzt werden, die aus puren Python-Skripten bestehen. Bibliotheken, die zusätzliche kompilierte Objekte benötigen, werden aufgrund des Security-Konzepts nicht unterstützt.

Wenn eine Python-Bibliothek genutzt werden soll, die aus puren Python-Skripten besteht, muss diese in die passenden Suchpfade (siehe Abschnitt "Suchpfade für Python-Module") kopiert werden. Das kann z. B. über „App-Daten verwalten“ erfolgen, siehe https://docs.automation.boschrexroth.com/doc/820023435/fenster-app-daten-verwalten/latest/de/. Wenn eine Python-Bibliothek auf einer ctrlX CORE genutzt werden soll, dann kann eine komplette Python-Laufzeitumgebung (inklusive aller notwendigen, kompilierten Objekte) in eine eigene App integriert werden. In diesem Fall muss die ctrlX MOTION per Data Layer kommandiert werden bzw. Data per Data Layer abgefragt werden.

Es empfiehlt sich, dafür REST-Aufrufe zu nutzen (z. B. mit der Requests-Bibliothek, https://requests.readthedocs.io/en/latest/). Die notwendigen Daten (sowohl beim Kommandieren, als auch beim Abfragen von Zuständen) werden als JSON Objekte genutzt. Die integrierte Bibliothek json (https://docs.python.org/3/library/json.html) hilft bei der Arbeit mit diesen Daten.

Minimalbeispiel für REST-Aufrufe:

```
import requests
#IP address
ip_addr = "192.168.1.1"
# get bearer token
bearer_addr = "https://"+ip_addr+"/identity-manager/api/v2/auth/token"
command_data = {"name":"boschrexroth","password":"boschrexroth"}
res = requests.post(bearer_addr, json=command_data, verify=False)
token = res.json()["access_token"]

# send command
res = requests.post('https://' + ip_addr + '/automation/api/v2/nodes/motion/axs/' +"Axis1" +'/cmd/pos-abs',
                json={"type":"object",
                    "value":{"axsPos":10,
                            "buffered":False,
                            "lim":{"vel":10,
                                "acc":10,
                                "dec":10,
                                "jrkAcc":0,
                                "jrkDec":0}}},
                headers={ 'Authorization': 'Bearer ' +
                        token },
                verify=False)
#get CmdId
cmdId = res.json()["value"]
print(cmdId)
```

Suchpfade für Python-Module

Der Konfigurationspfad für den Skriptmanager wird in $SNAP_COMMON (/var/snap/rexroth-automationcore/common) durch die App "rexroth-automationcore" bereitgestellt.

Python-Skripte ohne relative oder absolute Pfadangabe werden in $SNAP_COMMON/solutions gesucht.

Skriptausführung

Skriptausführung

Für die Instanz "robot" soll das Skript $SNAP_COMMON/solutions/activeSolution/script/loadWorkpiece.py abgearbeitet werden.

Unter dem Data Layer Knoten *script/instances/robot/cmd/file* wird dazu der Payload {"name":"activeSolution/script/loadWorkpiece.py","param":"pallet1"} gepostet.

Alternativ ist die absolute Pfadangabe mit "/var/snap/rexroth-automationcore/common/solutions/activeSolution/script/loadWorkpiece.py" möglich.

Importierte Python-Module werden in folgender Reihenfolge gesucht:

Aktuelles Skriptverzeichnis

- ./

Anwendungsmodule:

- $SNAP_COMMON/solutions/activeConfiguration/scripts/user
- $SNAP_COMMON/solutions/activeConfiguration/scripts/oem
- $SNAP_COMMON/solutions/activeConfiguration/scripts/bosch

Bibliotheken, wenn diese zum Zeitpunkt der Instanzerzeugung vorhanden sind:

- $SNAP_COMMON/solutions/activeConfiguration/scripts/libraries/user
- $SNAP_COMMON/solutions/activeConfiguration/scripts/libraries/oem
- $SNAP_COMMON/solutions/activeConfiguration/scripts/libraries/bosch