Table of contents

Python runtime environment

Python runtime environment

Python Runtime App for ctrlX CORE - Basics

The Python Runtime app contains a Python interpreter that has been specifically extended to interact highly efficiently with ctrlX MOTION. Thus, scripts specifying motion commands can be processed and can react to the state of ctrlX MOTION.

Script control via the script manager

A generic script manager is integrated in ctrlX CORE. The script manager can create and control script interpreter instances. A description of the script manager can be found in the ctrlX CORE Runtime application manual documentation, chapter Script parser/interpreter.

To run a Python script in the Python runtime environment, these steps have to be executed:

1. Copy the script into the solution (see section "Search paths for Python modules")
2. Creating a Python interpreter instance via script manager
3. Run script in this instance (see section "Script execution")

   (The Python interpreter instance can be used repeatedly and only needs to be created once).

Integrated libraries

Two libraries are integrated into the Python runtime environment:

- motion - Specifying commands and reading out of ctrlX MOTION states, see ↘ “Integrated library - motion”
- datalayer - Simple access to the Data Layer, see ↘ “Integrated library - datalayer”
  In addition, all Python libraries that are integrated in Python by default (e.g. sys) are available.

Troubleshooting

The error handling of the two libraries is realized via Python exceptions. Each time a function call fails, an exception is triggered (typically a RuntimeError). This exception can be recorded and the user can use individual troubleshooting. If the exception is not recorded, the script is canceled and all attached (see motion.attach_obj()) kinematics and axes are stop (with a minor error).

Name arguments

The name arguments in the functions of both libraries can be used on demand. All named arguments can be used an unnamed arguments (the sequence is relevant).

Additional Python libraries

Only Python libraries consisting of pure Python scripts can be used in the Python runtime environment. Libraries that require additional compiled objects are not supported due to the security concept.

If a Python library consisting of pure Python scripts is to be used, it has to be copied to the appropriate search paths (see section "Search paths for Python modules"). This can be done, for example, via "Manage app data", see https://docs.automation.boschrexroth.com/doc/820023435/window-manage-app-data/latest/en/. If a Python library is to be used on a ctrlX CORE, a complete Python runtime environment (including all required compiled objects) can be integrated into a customized app. In this case, the ctrlX MOTION has to be commanded via Data Layer or data has to be queried via Data Layer.

It is recommended to use REST calls for this purpose (e.g. with the Requests library, https://requests.readthedocs.io/en/latest/). The required data (both for commanding and for querying states) are used as JSON objects. The integrated json library (https://docs.python.org/3/library/json.html) facilitates working with this data.

Minimal example for REST calls:

```
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)
```

Search paths for Python modules

The configuration path for the script manager is provided in $SNAP_COMMON (/var/snap/rexroth-automationcore/common) by the "rexroth-automationcore" app.

Python scripts without relative or absolute path specification are searched in $SNAP_COMMON/solutions.

Script execution

Script execution

For the "robot” instance, process the $SNAP_COMMON/solutions/activeSolution/script/loadWorkpiece.py script.

Under the Data Layer node *script/instances/robot/cmd/file*, the payload {"name":"activeSolution/script/loadWorkpiece.py","param":"pallet1"} is posted.

Optionally, the absolute path can be specified using "/var/snap/rexroth-automationcore/common/solutions/activeSolution/script/loadWorkpiece.py".

Imported Python modules are searched in the following order:

Current script directory

- ./

Application modules:

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

Libraries if available at the time of instance generation:

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