Table of contents

Script parser/interpreter (Python)

Script parser/interpreter (Python)

Introduction and overview

Scripts can be executed with the ctrlX CORE. Scripts are simple text files with commands in the respective script language. The commands of the script file are subsequently processed during the execution. Very simple and intuitive command sequences can be created.

To execute a script, an app has to be installed with the respective script language. The ctrlX AUTOMATION currently provides the script language Python. More script languages are planned. Each script language is extended by specific functions for ctrlX CORE. Their specific syntax is described in the functions/commands.

All scripts are part of the configuration and have to be stored in the “Solution” to be processed in the ctrlX CORE.

User interface via Data Layer

To process a script, create the script interpreter instance first. Specify the script language in which the instance is to be processed. Any number of script interpreter instances can be created (limited by the ctrlX CORE memory). The instances run completely independent from each other.

After a script interpreter instance has been created, an execution script can be specified. The script starts automatically.

To query information on the current state, use the interfaces in the ↘ data layer A sequence control is also possible. It can be used to cancel the script processing for example.

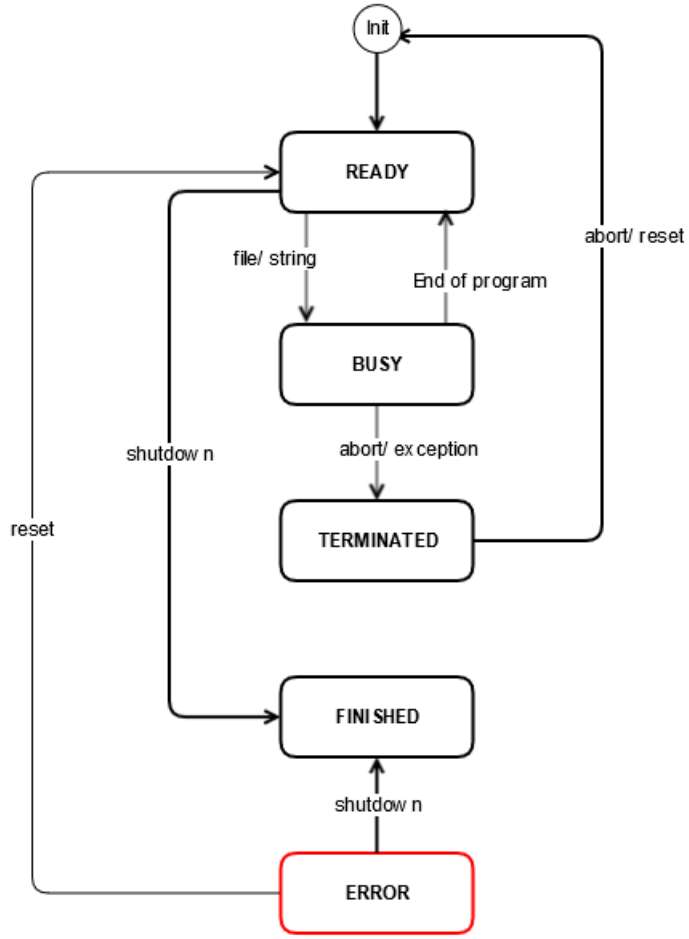

Fig. 34: Sequential control

Initialization script

An initialization script can be configured in the ctrlX CORE. When starting the ctrlX CORE, a script interpreter instance is created automatically and the configured script is started in this instance.

This script can for example:

- Creating additional script interpreter instances
- Starting more scripts in the generated script interpreter instances
- Writing a configuration into the Data Layer
- Executing first commands
- etc.

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

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 specification can be realized 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

Python functions in the Data Layer

Function (nodes) = datalayer.browse (<path>)

(nodes) = datalayer.browse (<path>)

The function browses through a specified Data Layer path and returns all nodes of a level below this path.

- <path>
  String, Data Layer path to be browsed through
- (nodes)
  Tuple of node names

Function <value> = datalayer.read (<path>)

<value> = datalayer.read (<path>)

The function reads an element of the Data Layer and returns its value. Only simple types are supported.

- <path>
  String, Data Layer path to be read
- <value>
  Value node (only simple values are supported)

Function datalayer.write(<path>, <value>)

datalayer.write(<path>, <value>)

The function writes an element of the Data Layer. Only simple types are supported.

- <path>
  String, Data Layer path to be written
- <value>
  Value to be written (only simple values are supported)

Function datalayer.create(<path>, <value>)

datalayer.create(<path>, <value>)

The function creates an element with the specified value in the Data Layer. Only simple types are supported.

- <path>
  String, Data Layer path to be created
- <value>
  Value node (only simple values are supported)

Function datalayer.remove(<path>)

datalayer.remove(<path>)

The function deletes an element of the Data Layer.

- <path>
  String, Data Layer path to be deleted

Function datalayer.read_json(<path>)

datalayer.read_json(<path>)

The function reads an element of the Data Layer and returns its value as JSON string.

- <path>
  String, Data Layer path to be read
- <value>
  Value of the node as JSON string

Function datalayer.write_json(<path>, <value>)

datalayer.write_json(<path>, <value>)

The function writes an element of the Data Layer. It is specified as JSON string.

- <path>
  String, Data Layer path to be written
- <value>
  Value to write the JSON string

Function datalayer.create_json(<path>, <value>)

datalayer.create_json(<path>, <value>)

The function creates an element with the specified value in the Data Layer. The value is a JSON string

- <path>
  String, Data Layer path to be created
- <value>
  Value of the node JSON string
- <result>
  Result during the creation as JSON string (only if successful)

Introduction and overview

Introduction and overview

Scripts can be executed with the ctrlX CORE. Scripts are simple text files with commands in the respective script language. The commands of the script file are subsequently processed during the execution. Very simple and intuitive command sequences can be created.

To execute a script, an app has to be installed with the respective script language. The ctrlX AUTOMATION currently provides the script language Python. More script languages are planned. Each script language is extended by specific functions for ctrlX CORE. Their specific syntax is described in the functions/commands.

All scripts are part of the configuration and have to be stored in the “Solution” to be processed in the ctrlX CORE.

User interface via Data Layer

User interface via Data Layer

To process a script, create the script interpreter instance first. Specify the script language in which the instance is to be processed. Any number of script interpreter instances can be created (limited by the ctrlX CORE memory). The instances run completely independent from each other.

After a script interpreter instance has been created, an execution script can be specified. The script starts automatically.

To query information on the current state, use the interfaces in the ↘ data layer A sequence control is also possible. It can be used to cancel the script processing for example.

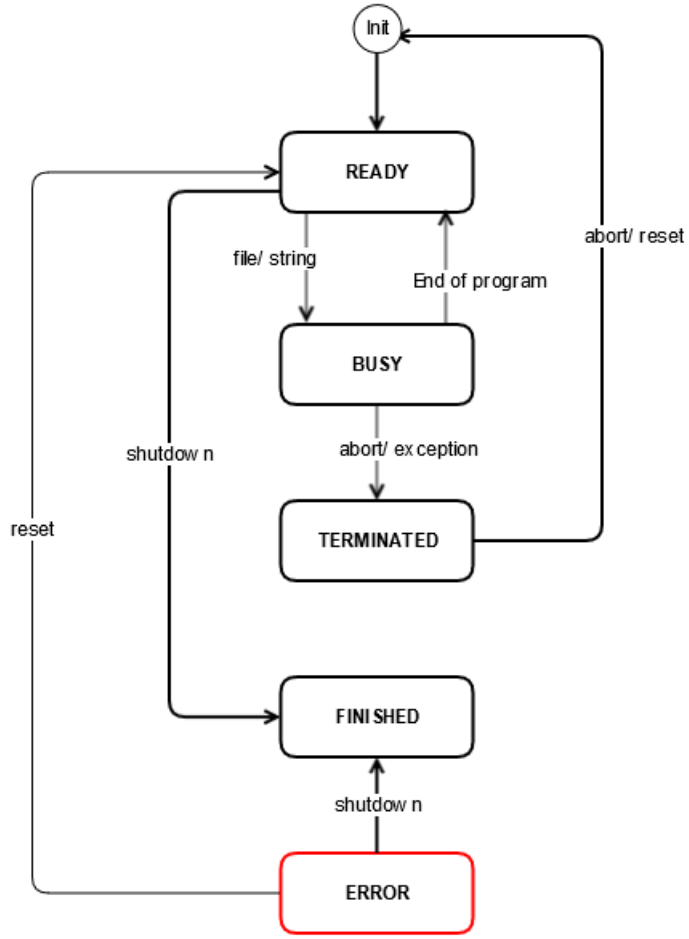

Fig. 34: Sequential control

Initialization script

Initialization script

An initialization script can be configured in the ctrlX CORE. When starting the ctrlX CORE, a script interpreter instance is created automatically and the configured script is started in this instance.

This script can for example:

- Creating additional script interpreter instances
- Starting more scripts in the generated script interpreter instances
- Writing a configuration into the Data Layer
- Executing first commands
- etc.

Search paths for Python modules

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

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 specification can be realized 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

Python functions in the Data Layer

Function (nodes) = datalayer.browse (<path>)

Function (nodes) = datalayer.browse (<path>)

(nodes) = datalayer.browse (<path>)

The function browses through a specified Data Layer path and returns all nodes of a level below this path.

- <path>
  String, Data Layer path to be browsed through
- (nodes)
  Tuple of node names

Function <value> = datalayer.read (<path>)

Function <value> = datalayer.read (<path>)

<value> = datalayer.read (<path>)

The function reads an element of the Data Layer and returns its value. Only simple types are supported.

- <path>
  String, Data Layer path to be read
- <value>
  Value node (only simple values are supported)

Function datalayer.write(<path>, <value>)

Function datalayer.write(<path>, <value>)

datalayer.write(<path>, <value>)

The function writes an element of the Data Layer. Only simple types are supported.

- <path>
  String, Data Layer path to be written
- <value>
  Value to be written (only simple values are supported)

Function datalayer.create(<path>, <value>)

Function datalayer.create(<path>, <value>)

datalayer.create(<path>, <value>)

The function creates an element with the specified value in the Data Layer. Only simple types are supported.

- <path>
  String, Data Layer path to be created
- <value>
  Value node (only simple values are supported)

Function datalayer.remove(<path>)

Function datalayer.remove(<path>)

datalayer.remove(<path>)

The function deletes an element of the Data Layer.

- <path>
  String, Data Layer path to be deleted

Function datalayer.read_json(<path>)

Function datalayer.read_json(<path>)

datalayer.read_json(<path>)

The function reads an element of the Data Layer and returns its value as JSON string.

- <path>
  String, Data Layer path to be read
- <value>

Value of the node as JSON string

Function datalayer.write_json(<path>, <value>)

Function datalayer.write_json(<path>, <value>)

datalayer.write_json(<path>, <value>)

The function writes an element of the Data Layer. It is specified as JSON string.

- <path>
  String, Data Layer path to be written
- <value>
  Value to write the JSON string

Function datalayer.create_json(<path>, <value>)

Function datalayer.create_json(<path>, <value>)

datalayer.create_json(<path>, <value>)

The function creates an element with the specified value in the Data Layer. The value is a JSON string

- <path>
  String, Data Layer path to be created
- <value>
  Value of the node JSON string
- <result>
  Result during the creation as JSON string (only if successful)