Adapter processes

General overview

Adapter processes are process that are responsible for interfacing and controlling various hardware components in the warehouse so that the main processes are not concerned with hardware related communication. From SRC side, these hardware components comprise PLC stations on the prezone, as well as shuttles and lifts in the OSR. Therefore, in SRC we have shuttle related adapter processes and a PLC adapter.

Shuttle adapters

For shuttle control, there are various processes, depending on the type of OSR. For OSR types before the Evo system, there is just the ***shuttle*** process which directly connects from the ***spider*** process to the physical shuttles. With Evo OSRs, there is an additional ***grid*** process which sits between ***spider*** and **shuttle**. *And with an Evo 2D system, there is also a* **cruise_control*** process which ensures that there are no collisions between shuttles on the same level.

Configuration

For SRC Commissioners, there is not so much to do in terms of shuttle configuration – most parts will be done by Shuttle Commissioners. What you need to do basically is ensuring that there all used shuttles are listed in the configurations, and that the hostnames of the shuttles are configured correctly so that they match the network settings on the Linux side; otherwise, the shuttle process will not be able to reach the shuttles on a network level. When working with a template config during preparation, you need to remove any instances of the “!TODO!” string in the shuttle config as well.

PLC adapter

The PLC adapter is responsible for connecting to the PLC CPUs in the warehouse. There will be as many ***plc*** processes as there are CPUs; still, there is only one config file.

Configuration

In the plc adapter config file, each and every PLC station that SRC needs to communicate to must be listed – with its numerical ID (the station numbering scheme is documented in the config file) and its type. Also, every PLC CPU must be configured (with its hostname so that it can be reached over the network), and for each CPU you need to configure which stations it is responsible for.

plc adapter

Christian Bretterklieber

v 1.1

Table of Contents

Overview

General

plc is the adapter process that connects the SRC system with the physical PLC CPUs, and which is responsible for all the communication with the PLC stations. Therefore, this process implements the low level communication with the hardware stations and thus represents the lowest level in a hierarchical view of the processes.

It is important to note that there can be more than one plc service in an SRC environment – actually, there will be one process per physical PLC CPU. Generally, the processes should be named according to the CPU name, if there is more than one; e.g. "***plca***" for *CPU A*, "***plcb***" for *CPU B*, and so on.

Physical PLC setup

The PLC ("*Programmable Logical Controller*") is a system that has been designed and adapted for industry environments; where its main purpose is to control lots of input/output devices. At KNAPP, there is a specific commissioning team for PLC, so for our purposes a high-level understanding of the PLC is sufficient.

PLC CPU

A PLC CPU is the processor unit which controls several PLC stations. It interprets the incoming signals from the hardware units, executes the control program, and sends output messages to the devices. The PLC CPUs and its components are housed in a switch cabinet which will be found near the OSR system (or the conveyor system, respectively).

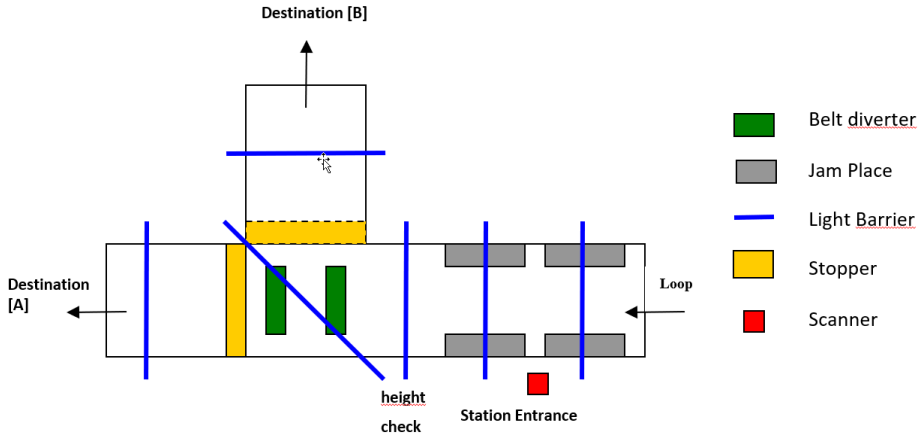
PLC switch cabinet

PLC station

A PLC station is a set of PLC controlled hardware which forms a logical unit with which we communicate – SRC receives incoming information, and sends commands to this station. On the actual PLC side, the PLC CPU is responsible then for triggering all required actions across the respective devices.

A typical PLC station in the prezone (e.g. fill reader or diversion to a picking station will usually consist of the following physical components:

- one or more light gates (one may be specifically used for a height control)
- a barcode reader to read the number of the passing container
- possibly a scale for weight checks
- a belt diverter to allow for a container to leave the loop



PLC station

Station numbering in SRC

Each and every PLC station has a unique station ID within the SRC system (the SRC ID for a PLC station has no impact on the PLC itself). In general, the station numbers can be arbitrarily chosen, but the convention is to use standardized station numbers. In the recent software revisions, we use 5-digit station numbers, according to the following structure:

- *first digit*: denotes the floor of the building where the station is located (e.g. "*1*" is ground floor, "*2*" is mezzanine,…)
- *second digit*: identifies the level of the conveyor (e.g. "*1*" is target conveyor line, "*2*" is the first source conveyor loop)
- *third digit*: describes the station type – see below
- *digits 4 & 5*: sequential counter of the stations, starting with "*00*"

The convention for numbering PLC stations responsible for lifts is a little different – here the first digit specifies the side that the lift is on ("*1*" is left/West, "*2*" is right/East) within its aisle, and the second digit is always a "*1*".

For the sequential numbering, the convention for stations pertaining to OSR aisles is to start at aisle 1 (or whatever the lowest number of the aisles is), and for diversions to picking stations to start at the picking station with the lowest number.

The station numbers should be documented in the skeleton plan (and / or the station communication document), and it is important that the station numbers are adhered to as they need to match what the PLC commissioning team configures on their side.

Station types

Below is a list of the most common station types used in the station numbering:

- "**4**": ("helper") PLC station at the prezone (loop switch or diverter)
- "**5**": diversion to an aisle
- "**6**": lift station
- "**7**": diversion to a picking station
- "**8**": picking station
- "**9**": lift entrance reading

Examples

Based on the explanations above, below are some examples of station numbers which you would regularly find at a project:

- "***12400***": first loop switch on source loop on ground floor
- "***11700***": diversion to first picking station on the target conveyor
- "***12700***": diversion to first picking station on the source loop
- "***12501***": diversion to aisle 2 on the source loop
- "***11600***": first lift on the left side of the aisle
- "***21601***": second lift on the right side of the aisle

Configuration

General

The plc configuration file (usually named "*plc_adapter_cfg.py*") is located in the `cfg` directory of the OSR instance. Please note that there is only one configuration file, regardless how many plc processes (depending on the number of PLC CPUs) there are.

As usual, at the beginning of the config file the required libraries are imported, then there will usually be a helper function for creating the plc stations (instances of the `OSR.PlcAdapterCfg.StationCfg` class.)

The main part – where you will have to do your config changes – is then the definition of all the PLC stations.

For each and every PLC CPU is then the class `OSR.PlcAdapterCfg.Config` instantiated (with a list of all station IDs that belong to the respective CPU), and finally the configurations of both the PLC (CPU) adapters and the PLC stations are written to the GCS, and the function to historize the configuration is called.

PLC stations and PLC configs

PLC stations

The example below shows the handling of creating stations for 2 different CPUs – one CPU is solely responsible for stations within the OSR (i.e. the lifts and lift entrance readings), the other one is taking care of all the other PLC stations.

```
def createPlcStation(id, type, descr, grp):
   return OSR.PlcAdapterCfg.StationCfg(
   stationId = id,
   type = type,
   containerFormat = "",
   destMap = [],
   description = "%s" % descr,
   group = grp
)

stationsPlcA= [
    createPlcStation(12900, OSR.PlcAdapterCfg.SimpleStation, "Aisle 1 Entrance",
"LIFTS"),
```

```
    createPlcStation(11600, OSR.PlcAdapterCfg.EvoLift, "Aisle 1 Lift 1", "LIFTS"),
    createPlcStation(21600, OSR.PlcAdapterCfg.EvoLift, "Aisle 1 Lift 2", "LIFTS"),


    createPlcStation(9999, OSR.PlcAdapterCfg.SafetyAreaStation, "PLC Logging
Station", "MESSAGE-STATION"),
]

stationsPlcB = [
    createPlcStation(12500, OSR.PlcAdapterCfg.SimpleStation, "Decision: Aisle 1 or
Loop",                    "LOOP-AISLES"),

    createPlcStation(12700, OSR.PlcAdapterCfg.SimpleStation, "Source Decision: PS21
or Loop", "LOOP-PICKSTATIONS"),
    createPlcStation(12701, OSR.PlcAdapterCfg.SimpleStation, "Source Decision: PS22
or Loop", "LOOP-PICKSTATIONS"),


    createPlcStation(9998, OSR.PlcAdapterCfg.SafetyAreaStation, "PLC Logging
Station", "MESSAGE-STATION"),
]
```

As you can see from the examples above, there are several parameters in the function call to instantiate the `OSR.PlcAdapterCfg.StationCfg` class:

- the station ID: refer to the station numbering scheme above
- the station type: there are some pre-defined station type classes that can be used here (like `OSR.PlcAdapterCfg.SimpleStation` for most “regular” stations, or `OSR.PlcAdapterCfg.EvoLift` for lift stations); you can find all different station types (and a short explanation) in the “*plc_adapter_cfg.idl*” file within the `~/knapp/lager/idl` directory
- a description of the station
- a logical group to which you can assign the PLC station

The description and the logical group are relevant for the SCADA warehouse visualization software.

PLC configs

The code below is a continuation of the above configuration – the configurations for two PLC CPUs are done by calling a helper function which instantiates the `OSR.PlcAdapterCfg.Config` class:

```
def plc_config (station, hostname):
    return OSR.PlcAdapterCfg.Config(
        stations = station,
        plcIpAddress = hostname,
        localTSAP = "\xE0\x02\x33",
```

```
        remoteTSAP = "\xE0\x02\x33",
        tlvCatLogLevel = OSR.LL_INFO,
        maxPacketsPerTSDU = 5,
        maxBytesPerTSDU = 1000,
        maxBytesPerTPDU = 0,
        reconnectInterval = 5000,
        acknowledgementTimeout = 2000,
        heartBeatInterval = 10000,
        plcPort = 102,
        sendSpeedAndAccelerationToLift = False)

...

stationIdsPlcA =  [x.stationId for x in stationsPlcA]
stationIdsPlcB =  [x.stationId for x in stationsPlcB]

...


try:
    osr_ctx.unbind(["plc_adapterA"])
    osr_ctx.unbind(["plc_adapterB"])
except:
    pass

osr_ctx.bind(
    ["plc_adapterA"],
    CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/Config:1.0"),
    plc_config(stationIdsPlcA, "osrplca"))
)

osr_ctx.bind(
    ["plc_adapterB"],
    CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/Config:1.0"),
    plc_config(stationIdsPlcB, "osrplcb"))
)
```

In this example, there are first two Python lists (`stationIdsPlcA` and `stationIdsPlcB`) created which just hold the IDs of the PLC stations; in this case, the lists are constructed using a Python syntax called "*list comprehension*" – another way would be to manually populate the lists.

Following that, the PLC adapter configs are constructed for two PLC CPUs, and made available to GCS through CORBA. Here you have to do the following:

- First you call the function `osr_ctx.unbind()` to remove any bindings to configuration names (in our case "***plc_adapterA***" and "***plc_adapterB***"). These names can be arbitrarily chosen, but they need to match the names that are used in the plc processes' run files.
- Then you bind these configuration names to the configurations of the PLC CPUs. When constructing the configuration class, you need to use the following arguments:
  - the respective list of the corresponding PLC stations ("*stationIdsPlcA*" and "*stationIdsPlcB*")
  - the hostname of the respective PLC CPU (this hostname must be verified against the entries in `/etc/hosts`)

Process startup

Similar to the other processes of SRC, the plc process is started through a wrapper `run` shell script which is located underneath the plc's subdirectory within the `_service` folder of the OSR instance. If there is more than one plc process required, you will have to create several directories (like `plca`, `plcb`, and so on), and configure the respective run files.

Within the run file, at the actual call of the process, you need to configure the correct PLC adapter name, as configured in the config file:

```
exec jaco -XX:-OmitStackTraceInFastThrow \
       -Dosr-id=${OSR_ID} \
       -DOAPort=$port \
       -Dlog4j2.configurationFile=$OSR_HOME/inst/$OSR_ID/cfg/log4j.xml \
       -Dlog4j2.enable.threadlocals=false \
       -Djacorb.connection.client.pending_reply_timeout=5000 \
       at.syslog.osr.adapters.plc.Main plc_adapterA true 2>&1
```

PLC communication

For the general message flow in the communication between the plc adapter and SRC see the relevant section in the steuer handout.

Below you can find some information about flags that are communicated by the PLC towards SRC when a container is clamped:

- **UNK**: unknown container
- **RE**: reading error; barcode was not correctly read
- **HF**: hardware failure
- **SF**: “station full”; diversion not possible as the segment after the diversion is filled
- **OOW**: “out of window” – the PLC station has timing issues
- **OH**: over-height: the container triggered the height control

Version information

While we strive to keep the training documentation as generic as possible, there will be differences across platforms and SRC software revisions, concerning e.g. directory structure or code syntax. Below you can find an explanation as for which version(s) and platform(s) the documentation is mostly relevant, as well as a version history.

This is **version 1.1** of the document.

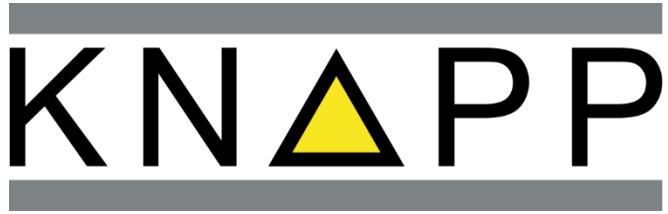
Version relevance

This information in this document has been created focusing on the following versions – the first number is the most relevant one; versions in parentheses are expected be applicable, but were not tested specifically:

- Platform: 13.1, (12)
- SRC: 6.3, (6.2)

Version history

Version	Description
1.1	Minor changes
1.0	Initial version



Adapter processes

What are adapter processes?

Adapter processes in a nutshell

PURPOSE

- SRC needs to interface with and control different hardware components:
  - PLC stations in the prezone
  - Lifts in the OSR
  - OSR shuttles
- In order to decouple hardware oriented communication from the main processes, SRC utilizes adapter processes to communicate with the hardware components
  - plc adapter process(es) – there will be one process per PLC CPU
  - Shuttle related processes:
    - shuttle
    - grid
    - (cruise_control)

Important terms

Terms

- **PLC station**
  - Group of hardware components controlled by PLC as logical unit
- **PLC CPU**
  - Central controller unit for PLC stations; situated in a cabinet

Shuttle processes

Shuttle adapter interfaces

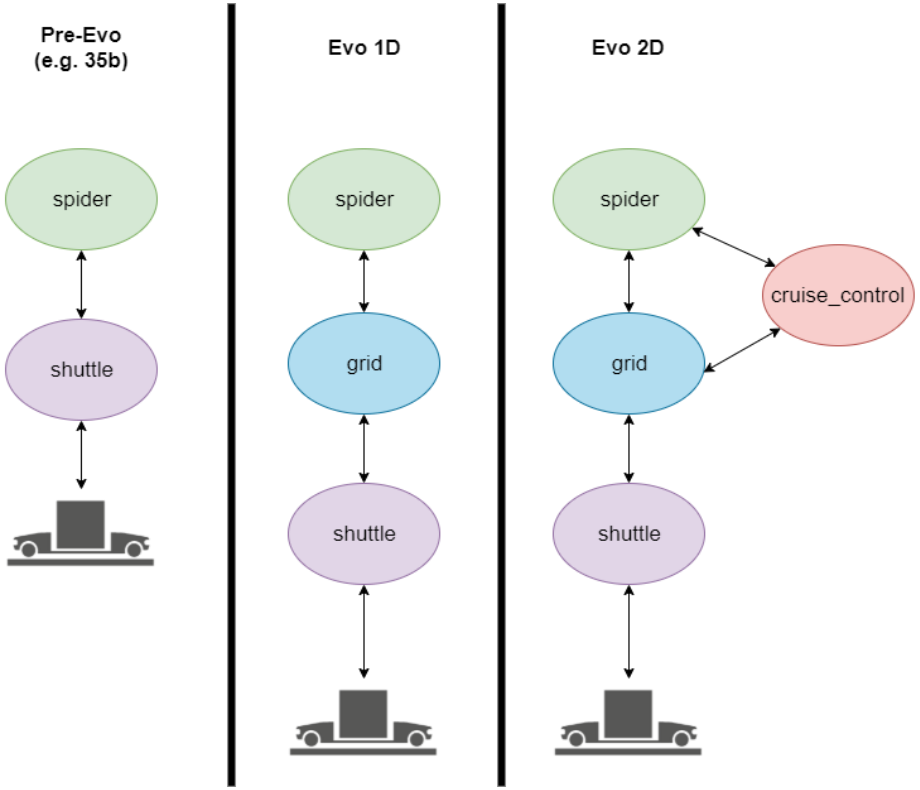
DEFINITION

Connection between SRC and the shuttles

- pre Evo: spider ↔ shuttle (process) ↔ shuttles
- Evo: spider ↔ grid ↔ shuttle adapter ↔ shuttles

shuttle adapter connecting to shuttles

- TCP/IP connection, port 2101



Communication logs

Shuttle adapters

Shuttle logs – software-side

EXAMPLE

```
spider  2021-06-25 02:16:02.437 INFO  [SHUT0057] Starting activity: { transporter: SHUT0057, MOVE_TO_LOC ( S03.019.0070.
        2021-06-25 02:16:02.437 INFO  [SHUT0057_transport] moveToLocation({ transporter: SHUT0057, MOVE_TO_LOC ( S03.019
        2021-06-25 02:16:02.437 INFO  [SHUT0057_transport] <-- drive(path: S03.019.0070.L, boxWidth: 400, checkRack: fal

grid    2021-06-25 02:16:02.437 INFO  [SHUT0057] --> drive(path: [S03.019.03.01.0001.R, S03.019.03.01.0070], boxWidth: 4
        2021-06-25 02:16:02.441 INFO  [SHUT0057_delegate] --> startDrive(orderId: 14431, boxWidth: 400, accelerationFact

shuttle 2021-06-25 02:16:02.442 INFO  [SHUT0057] --> startDrive(orderId: 14431, targets: [(S03.019.03.01.0001.R, speed:
        2021-06-25 02:16:15.281 INFO  [SHUT0057] <-- finishedDrive(orderId: 14431, position: S03.019.03.01.0070, deviati

grid    2021-06-25 02:16:15.282 INFO  [SHUT0057_delegate] <-- finishedDrive(position: S03.019.03.01.0070, deviation: 1,
        2021-06-25 02:16:15.282 INFO  [SHUT0057] <-- drove(shuttleStatus: GridShuttleStatus(busy: false, reservations: [

spider  2021-06-25 02:16:15.283 INFO  [SHUT0057_transport] drove(status: { curPos: S03.019.03.01.0070, idle, referenced:
        2021-06-25 02:16:15.283 INFO  [SHUT0057] Activity finished [  12846ms]: { transporter: SHUT0057, MOVE_TO_LOC ( S
```

Shuttle logs – interface side

EXAMPLE

```
2021-06-25 02:16:02.442 INFO  [SHUT0057] --> startDrive(orderId: 14431, targets: [(S03.019.03.01.0001.R, speed:
2021-06-25 02:16:02.442 DEBUG [SHUT0057] Sending Telegram: GetGeneralStatusTelegram: (version: 1, adr: 1, msgID:
2021-06-25 02:16:02.445 DEBUG [SHUT0057] Received telegram: GeneralStatusV3Telegram: (version: 1, adr: 1, msgID:
2021-06-25 02:16:02.445 DEBUG [SHUT0057] Sending Telegram: GetPositionStatusTelegram: (version: 1, adr: 1, msgI
2021-06-25 02:16:02.448 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgI
2021-06-25 02:16:02.449 INFO  [SHUT0057] <-- startingToDrive(transporterId: 300000057, positions: [(S03.019.03.0
2021-06-25 02:16:02.449 DEBUG [SHUT0057] Sending Telegram: MoveRackV2Telegram: (version: 1, adr: 1, msgID: 5, op
2021-06-25 02:16:02.452 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgI
2021-06-25 02:16:02.452 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0001.R, deviation: 1.
2021-06-25 02:16:02.496 INFO  [SHUT0057] --> updateDrive(orderId: 14431, targets: [(S03.019.03.01.0029, speed: 4
2021-06-25 02:16:02.497 INFO  [SHUT0057] <-- updatingDrive(transporterId: 300000057, positions: [(S03.019.03.01,
…
2021-06-25 02:16:07.518 DEBUG [SHUT0057] Sending response: AcknowledgeTelegram: (version: 1, adr: 1, msgID: -60,
2021-06-25 02:16:07.518 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0028, deviation: 1.
…
2021-06-25 02:16:15.281 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgI
2021-06-25 02:16:15.281 DEBUG [SHUT0057] Sending response: AcknowledgeTelegram: (version: 1, adr: 1, msgID: -55,
2021-06-25 02:16:15.281 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0070, deviation: 1.
2021-06-25 02:16:15.281 INFO  [SHUT0057] <-- positionStatusSupervisionAvailable(transporterId: 300000057, positi
2021-06-25 02:16:15.281 INFO  [SHUT0057] <-- finishedDrive(orderId: 14431, position: S03.019.03.01.0070, deviati
```

Configuration

Shuttle adapters

All examples taken from version 6.3

grid / shuttle configuration

DEFINITION

- From SRC Commissioning side, configuration for the shuttle adapter processes is not much; the majority of settings is done by Shuttle Commissioning
- Also, many settings are read by grid and shuttle during runtime from the warehouse model
- For preparation and use with simulator, some settings have to be done, though:
  - Number of shuttles
  - Hostnames / addresses for the shuttles (shuttle config)
  - File path to the warehouse model XML file
- In the grid adapter config, a rackAdmin config must be added for the PLC station(s) responsible for the shuttle states
- In the shuttle adapter config, there are many lines with “!TODO!” in front of them – this string needs to be removed to have the configuration file executable

grid configuration

CODE EXAMPLE

```
aisleCount = 2          # Number of aisles / towers
levelCount = 10         # Number of levels per aisle
startShuttleNr = 1      # Lowest shuttle number in this OSR instance
spareShuttleCount = 0   # Number of spare shuttles in this OSR instance

shuttleCount = (aisleCount * levelCount) + spareShuttleCount
aisleRange = range(1, aisleCount + 1)
levelRange = range(1, levelCount + 1)
shuttleRange = range(startShuttleNr, startShuttleNr + shuttleCount)

shuttleList = []
for shuttleId in shuttleRange:
#                       (URL, ID,     SEM-Mode)
    shuttleList.append(( shuttleId,  None ))

…
#  (transporter-ID, station-IDs)
rackAdminConfig = (  9001,  [9999, ])

…
xmlWarehouseModelPathname = "/home/osr/knapp/lager/inst/osr1/dat/wh_model.xml"
```

shuttle configuration

CODE EXAMPLE

```
aisleCount = 2          # Number of aisles / towers
levelCount = 10         # Number of levels per aisle
startShuttleNr = 1      # Lowest shuttle number in this OSR instance
spareShuttleCount = 0   # Number of spare shuttles in this OSR instance
semMode = 12            # 12 for SRC. 2 for Shuttle startup without grid-adapter
port = 2101

aisleRange = range(1, aisleCount + 1)
levelRange = range(1, levelCount + 1)
shuttleCount = (aisleCount * levelCount) + spareShuttleCount

deviceList = []
for shuttleNr in range(startShuttleNr, startShuttleNr + shuttleCount):
    url = "tcp://shuttle%04d:%d" % (shuttleNr, port)
#                       (URL, ID,         SEM-Mode)
    deviceList.append((url, shuttleNr, semMode ))

…
warehouseModelPath = "/home/osr/knapp/lager/inst/osr1/dat/wh_model.xml",  # Filepath of the used XML
```

shuttle configuration

CODE EXAMPLE

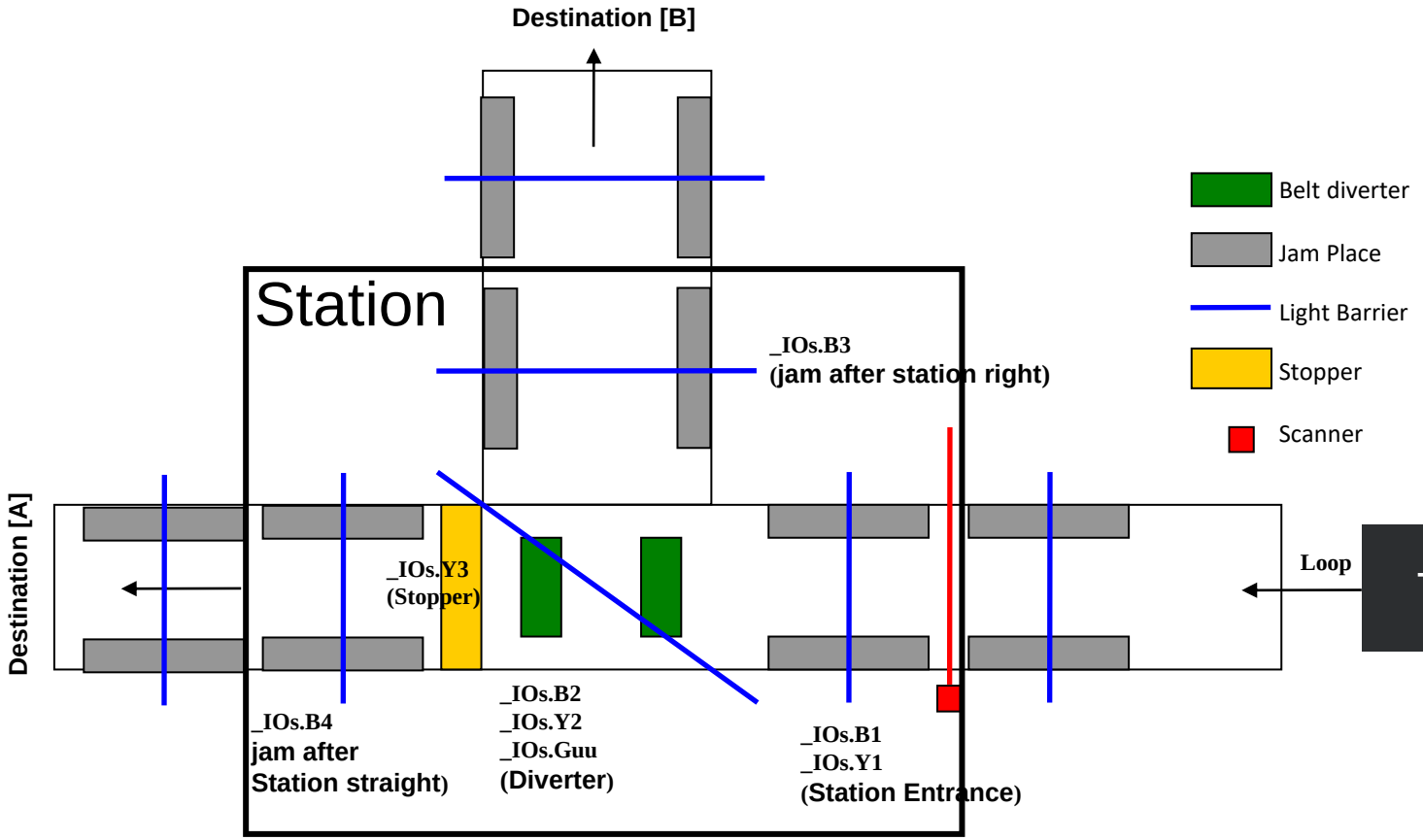
All entries of “!TODO!” need to be removed, otherwise config file cannot be executed!

```
def createDefaultShuttleConfig(param):
    return Shuttle35bManager.Shuttle35bConfig(
        shuttleName = "SHUTXXXXX",          # Name (ID) of the Shuttle -> configured individually
        uri = "tcp://0.0.0.0:0",            # TCP/IP address of the Shuttle -> configured individually
        communicationTimeout = 500000,      # The maximum amount of time, until a communication command has to be
!TODO!  positionTimeout = 500000,           # The maximum amount of time, until a positioning command has to be
        trayTimeout = 220000,               # The maximum amount of time, until a load/store command has to be re
        lhdTimeout = 40000,                 # The maximum amount of time, until a center command has to be replie
        positionUpdateTimeout = 15000,      # The maximum amount of time, until an EVO shuttle has to send positi
!TODO!  handlesChannels = 0,                # Whether the Shuttle is able to position to channels (used for OSR-S
!TODO!  hasVarioLhd = 0,                    # Shuttle-LHD with vario (0 = false, 1 = true)
        resendIntervalMillis = 10000,       # The amount of time to wait for a response, until the order will be
        maxSendCount = 5,                   # The amount of resents until the connection will be closed
        socketMaxReconnects = 5,            # The amount of reconnects to the Shuttle
        reconnectInterval = 2000,           # Waiting time between reconnects [ms]
        setCurrentTimeInterval = 3600000,   # Time-synchronization interval between Server and Shuttle [ms]
        incomingBufferSize = 16384,         # Buffer-size for enqueueing telegrams from the Shuttle
…
```

plc adapter process

PLC stations (prezone)

DEFINITION



Communication between PLC <-> Software

1. Clamped

```
[S1251] <-- containerClamped(CID: 50028557,
UNK: 1, RE: 0, HF: 0, SF: 0)
```

2. Announce

```
[S1251] -->announceContainer(CID:50028557,
SRC: 0, DEST: -1, HEIGHT: 1)
```

3. Processed

```
[S1251] <-- containerProcessed(CID:
50028557, POS: -1, UNK: 0, RE: 0, HF: 0, SF:
0, OOW: 0, OH: 0, OC: 0, NB: 0, TU: 0)
```

4. Delete

```
[S1251] --> deleteContainer(CID: 50028557)
```

Communication logs

PLC adapter

plc logs

EXAMPLES

Communication with a plc station on the loop:

```
2021-04-01 19:54:57.897 INFO  [S13704] <-- containerClamped(CID: 310012756, UNK: 1, RE: 0, HF: 0, SF: 0)
2021-04-01 19:54:57.900 INFO  [S13704] --> announceContainer(CID: 310012756, POS: 0)
2021-04-01 19:54:59.509 INFO  [S13704] <-- containerProcessed(CID: 310012756, POS: 0, UNK: 0, RE: 0, HF: 0, SF:
2021-04-01 19:54:59.510 INFO  [S13704] --> deleteContainer(CID: 310012756)
```

Communication with a lift:

```
2021-04-02 11:26:06.987 INFO  [S12602] --> moveToLevel(LEV: 11, LHD: 2)
2021-04-02 11:26:08.445 INFO  [S12602] <-- movedToLevel(LEV: 11, LHD: 2, ULvl: 0, ULhd: 0, HF: 0, EC: 0x0100000
2021-04-02 11:26:08.447 INFO  [S12602] --> load(LHD: 2|1, src LOC: 11|NORTH|1)
2021-04-02 11:26:09.688 INFO  [S12602] <-- loaded(dst LHD: 2|0, src LOC: 11|NORTH|1, NB: 0, RE: 0, LhdOcc: 0, NA
```

Configuration

PLC adapter

All examples taken from version 6.3

PLC stations / CPUs

DEFINITION

- PLC stations are groups of hardware components that work together at specific positions (e.g. a decision station for deviation of containers on the conveyor line) and are controlled by the PLC
  - On the conveyor lines, a PLC station usually consists of a barcode reader and a device for deviation of a container (e.g. transfer belt or roller switch)
  - PLC stations have a unique number – which must also match the configuration on PLC side
- Code implementation:
  - All PLC stations are configured as `PlcAdapterCfg.StationCfg` and written to the GCS
  - It is important to use the correct type of station as this determines the communication messages
  - For each PLC CPU an instance of `PlcAdapterCfg.Config` must be created and also published to the GCS

PLC stations

CODE EXAMPLE

```
def createPlcStation(id, type, descr, grp):
    return OSR.PlcAdapterCfg.StationCfg(
        stationId = id,
        type = type,
        containerFormat = "",
        destMap = [],
        description = "%s" % descr,
        group = grp
    )

stations = [
    # TrayLoop
    createPlcStation(12400, OSR.PlcAdapterCfg.SimpleStation, "Simple Station Entrance", "LOOP"),
    createPlcStation(12500, OSR.PlcAdapterCfg.SimpleStation, "Entrance Tower 1", "LOOP"),
…
    createPlcStation(12800, OSR.PlcAdapterCfg.TrayPickAreaStation, "Tray Pick Area", "LOOP"),
…
    #Tower Lifts
    createPlcStation(11600, OSR.PlcAdapterCfg.EvoLift, "Tower 1 Lift", "LIFTS"),
…
```

PLC CPUs

CODE EXAMPLE

```
stationIds = [x.stationId for x in stations]

def plc_config (station, hostname):
    return OSR.PlcAdapterCfg.Config(
        stations = station,
        plcIpAddress = hostname,
        localTSAP = "\xE0\x02\x33",
        remoteTSAP = "\xE0\x02\x33",
        tlvCatLogLevel = OSR.LL_INFO,
        maxPacketsPerTSDU = 8,
…
        maxMessageCount = -1)

…
osr_ctx.bind(["plc_adapter1"],
             CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/Config:1.0"),
                       plc_config(stationIds, "osrplca")))

…
osr_ctx.bind(["station_cfg_list"],
             CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/StationCfgList:1.0"),
                       stations))
```

Tips & Tricks

Tips & tricks

Tips

- Explore the IDL files for the configurations
  - Especially for checking out what station types exist for PLC stations
- Check out the inline documentation of the plc config for station numbering
  - The station numbering scheme has changed over time, so when working on older installations you may encounter different station numbers


KNAPP
making complexity simple

Shuttle related processes

Christian Bretterklieber

v 1.1

Table of Contents

Overview

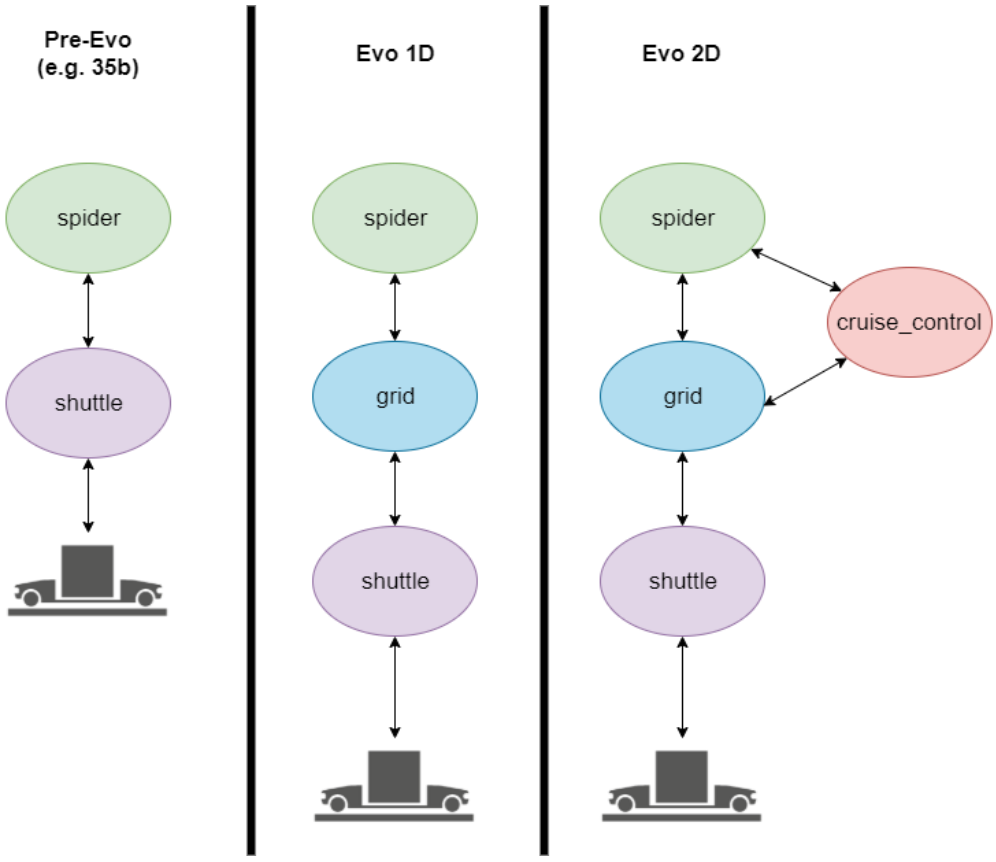
General

Depending on the version and type of the OSR, there will be a different amount of processes involved for controlling the OSR shuttles. In pre-Evo OSRs, there is only one process, the ***shuttle*** adapter process. In this case, there is a direct connection from spider to shuttle.

With the OSR Evo system, a new process – the ***grid*** adapter – has been introduced which sits between the spider and the shuttle process.

In addition, for OSR Evo 2D systems, there is yet another process which is called the ***cruise_control***. This process is responsible for double-checking that there will not be any collisions between shuttles.

The below diagram depicts the possible scenarios:



Possible scenarios for shuttle control

shuttle adapter

The shuttle adapter is the process that communicates directly with the shuttles in the OSR. This communication happens via TCP/IP on port 2101 – the shuttles are connected via WiFi in a separate network, and the shuttle process needs to know how to reach them; therefore it must have a list of all host names that are assigned to the shuttles through the Linux network configuration.

Data is exchanged between the shuttles and the shuttle adapter process in so-called "*telegrams*" – you will find this term in the log lines of the shuttle adapter log files.

As the shuttle adapter is the closest to the physical shuttle, its commands are the most direct ones – so it instructs the shuttle for instance about the exact distance (in mm) that it needs to drive, and when to perform a store or load action.

grid adapter

In Evo OSRs, the grid adapter process is an intermediate process – so there is no communication between spider and shuttle anymore. grid passes the commands it receives from spider down to shuttle; and for driving instructions, it breaks down the full distance of a required path into smaller slices which it sends to shuttle one after another. So for example, if spider sends to grid the instructions for a shuttle to load a tray at the buffer, and then store it to location 210 which is at the far end of the rack, grid may break this down into 3 slices which it sequentially forwards to shuttle.

Of course, the shuttle adapter continuously informs grid about the current position of the shuttle, so grid will send the next slice in the sequence early enough so that the shuttle does not stop during motion.

cruise_control

The cruise_control process is an additional process that was introduced for Evo 2D systems – it is basically a kind of "safety net" to ensure that there are no crashes between shuttles in the 2D system. It communicates with the spider and the grid process and monitors that the routes that that have been calculated do not lead to any collisions.

From Commissioning side, there is no specific configuration to be done for this process; there is no Python configuration, and only the corresponding run file must exist.

Configuration – shuttle

Network config

For the shuttle adapter to be able to communicate with the shuttles, they must have corresponding host names in the system which resolve to their IP addresses. Therefore, during commissioning, all shuttles must have corresponding entries in the `/etc/hosts` file. For real warehouses, this will be accomplished by doing a DHCP configuration where all shuttles' MAC addresses have to be entered and matched with the desired host name. The network configuration is beyond the scope of this document.

General

The config file for the shuttle process can be found as usual in the `cfg` folder of the OSR instance. It is important to note that the config file (usually "*shuttly_cfg.py*") will mostly be adapted by the Shuttle Commissioning team, and less so by SRC Commissining. Also, very importantly, there are many lines in the (template) config file which start with the string "*!TODO!*". If these strings are not removed, the config file cannot be executed. The reasoning behind is that some of these settings are potentially harmful as they could destroy equipment if not set correctly.

For in-house testing purposes (with using the simulator), it is ok to remove these strings and work with the default values, but onsite it is crucial to merge the Shuttle Commissioning's correct settings into the configuration before starting the shuttles.

As always, the top of the config file holds all import statements for required libraries. Then follow a few general settings. Somewhat different from most other configuration files in SRC, the main configuration class `Shuttle35bManager.Config` (yes, even with Evo the naming for some classes etc. is still "*shuttle35b*") is instantiated relatively early in the file; yet the shuttle specific settings are still being done afterwards, using a couple of helper functions.

At the end of the config file, as usual, the configuration is written to GCS and then historized.

Basic settings

As most of the shuttle specific parameters will be set by the Shuttle Commissioning team, there are only a few settings that need to be done; and for testing purposes with the simulator you also need to get rid of the "*!TODO!*" instances.

```
aisleCount = 3           # Number of aisles / towers
levelCount = 10          # Number of levels per aisle
startShuttleNr = 1       # Lowest shuttle number in this OSR instance
spareShuttleCount = 2    # Number of spare shuttles in this OSR instance
semMode = 12             # 12 for SRC. 2 for Shuttle startup without grid-adapter
port = 2101

shuttleCount = (aisleCount * levelCount) + spareShuttleCount
deviceList = []
for shuttleNr in range(startShuttleNr, startShuttleNr + shuttleCount):
   url = "tcp://shuttle%04d:%d" % (shuttleNr, port)
   #                 (URL, ID,        SEM-Mode)
   deviceList.append((url, shuttleNr, semMode ))
```

What you have to set here for testing is the `aisleCount` and `levelCount` variables. As you can see from the for loop, the host names for the shuttles are expected to be "*shuttle0001*" to "*shuttle0032*", so these entries will have to exist in the Linux network config (`/etc/hosts`).

General adapter config

```
cfg = Shuttle35bManager.Config(
   xPositionMappingTolerance = 70,          # Allowed mapping-tolerance between
physical values and generated values for the Shuttle position [mm]
   zPositionMappingTolerance = 70,          # Allowed mapping-tolerance between
physical values and generated values for the container position [mm]
   channelDepthMappingTolerance = 70,       # Allowed mapping-tolerance between
physical values and generated values for the channel-depth [mm]
   bootloaderReconnectDelay = 5000,         # Increase, if a firmware-download exits
with an PROTOCOL_ERROR [ms]
!TODO!    usesStaticOptimizedChannelStorage = 0,  # used, if the containers need to
be aligned on the front and back of each location in order to minimize the pressure
on the shelf
!TODO!    backAlignedLocationTypes = [],           # Location types for
StaticOptimizedChannelStorage (hmpLocationTypeName)
!TODO!    loadThreshold = 30,                      # The maximum current [A] which
can be used simultaneously per service-level (0 disables load-management)
   minTrayInterspace = 40,                  # Minimum distance between two
containers in the rack [mm] (z-direction)
 # This value is required to assure sufficient space for the flaps between the
containers
!TODO!    maxContainerRequiredPositions = 1,       # The number of logical z-
positions, which are used by the longest container in the system
   unknownContainerType = "0",              # The container type from the HMP-config
which is used for the unknown container
!TODO!    warehouseModelPath = "/home/osr/knapp/lager/inst/osr1/dat/wh_model.xml",
# Filepath of the used XML
   configs = [])
```

This is the overall configuration instance; the specific settings for all of the shuttles will be added later in the `configs` list). Most of the parameters here are relevant only for Shuttle Commissioning.

The important setting in this instance is the `warehouseModelPath` – this needs to be set to the correct path to the `wh_model.xml` file. If the path is not correct, the shuttle adapter will not work.

Configuration – grid

General

The “*grid_cfg.py*” file (also within the `cfg` folder) does not require much configuration effort either.

Basic settings

What is required to set here is the number of shuttles in the OSR – which needs to match the settings from the shuttle configuration.

```
aisleCount = 3           # Number of aisles / towers
levelCount = 10          # Number of levels per aisle
startShuttleNr = 1       # Lowest shuttle number in this OSR instance
spareShuttleCount = 2    # Number of spare shuttles in this OSR instance

shuttleList = []
for shuttleId in shuttleRange:
#                      (URL, ID,     SEM-Mode)
   shuttleList.append(( shuttleId,  None ))
```

Rack admin station

This setting defines the PLC station(s) responsible for controlling rack access. This is by default the station “*S9999*” – if there is more than one PLC CPU responsible for the OSR, the next number would by convention be “*S9998*”, and so forth.

```
#                  (transporter-ID, station-IDs)
rackAdminConfig = (      9001,       [9999, ])
```

Warehouse model path

Again, grid must be aware of the correct path to the wh_model.xml file for it to work correctly.

```
xmlWarehouseModelPathname = "/home/osr/knapp/lager/inst/osr1/dat/wh_model.xml"
```

Version information

While we strive to keep the training documentation as generic as possible, there will be differences across platforms and SRC software revisions, concerning e.g. directory structure or code syntax. Below you can find an explanation as for which version(s) and platform(s) the documentation is mostly relevant, as well as a version history.

This is **version 1.1** of the document.

Version relevance

This information in this document has been created focusing on the following versions – the first number is the most relevant one; versions in parentheses are expected be applicable, but were not tested specifically:

- Platform: 13.1, (12)
- SRC: 6.3, (6.2)

Version history

Version	Description
1.1	Minor changes
1.0	Initial version


KNAPP

shuttle adapter

types of shuttle adapter

shuttle adapter

for evo shuttles (1D/2D), s35b shuttles, newer s32 shuttles

shuttle classic adapter

for older s32 shuttles, s15 shuttles

interface process

spider/grid process connects to shuttle adapter

pre evo: spider <-> shuttle_adapter <-> shuttles

evo: spider <-> grid <-> shuttle_adapter <-> shuttles

shuttle adapter connects to shuttles

TCP/IP connection, port 2101

Configuration

template configs

shuttle adapter:

core/osr-6-2/java/at/syslog/osr/adapters/shuttle/shuttle_cfg.py
core/osr-6-2/java/at/syslog/osr/adapters/shuttle/shuttle_cfg_35b.py

shuttle classic adapter:

core/osr-6-2/java/at/syslog/osr/adapters/shuttle_classic/shuttle_manager_cfg.py

What to configure

number of shuttles

timeout settings

most other parameters (speed, acceleration, power settings,…):

default values for inhouse tests
real values configured by shuttle technicians

big parts of configuration are taken from wh_model.xml

```
aisleCount = 3          # Number of aisles / towers
levelCount = 10         # Number of levels per aisle
startShuttleNr = 1      # Lowest shuttle number in this OSR instance
spareShuttleCount = 2   # Number of spare shuttles in this OSR instance
semMode = 12            # 12 for SRC. 2 for Shuttle startup without grid-adapter
port = 2101

aisleRange = range(1, aisleCount + 1)
levelRange = range(1, levelCount + 1)
shuttleCount = (aisleCount * levelCount) + spareShuttleCount

deviceList = []
for shuttleNr in range(startShuttleNr, startShuttleNr + shuttleCount):
    url = "tcp://shuttle%04d:%d" % (shuttleNr, port)
#                       (URL, ID,        SEM-Mode)
    deviceList.append((url, shuttleNr, semMode ))
```

```
def createDefaultShuttleConfig(param):
    return Shuttle35bManager.Shuttle35bConfig(
        shuttleName = "SHUTXXXXX",          # Name (ID) of the Shuttle -> configured individually
        uri = "tcp://0.0.0.0:0",            # TCP/IP address of the Shuttle -> configured individually
        communicationTimeout = 500000,      # The maximum amount of time, until a communication command has to be replied [ms]
!TODO!  positionTimeout = 500000,           # The maximum amount of time, until a positioning command has to be replied [ms]
        trayTimeout = 220000,               # The maximum amount of time, until a load/store command has to be replied [ms]
        lhdTimeout = 40000,                 # The maximum amount of time, until a center command has to be replied [ms]
        positionUpdateTimeout = 15000,      # The maximum amount of time, until an EVO shuttle has to send position update when driving [ms]
!TODO!  handlesChannels = 0,                # Whether the Shuttle is able to position to channels (used for OSR-Shuttle 32) (0 = false, 1 = true)
!TODO!  hasVarioLhd = 0,                    # Shuttle-LHD with vario (0 = false, 1 = true)
        resendIntervalMillis = 10000,       # The amount of time to wait for a response, until the order will be resent [ms]
        maxSendCount = 5,                   # The amount of resents until the connection will be closed
        socketMaxReconnects = 5,            # The amount of reconnects to the Shuttle
        reconnectInterval = 2000,           # Waiting time between reconnects [ms]
```

!TODO! parameters

to ensure that config isn‘t executed with wrong default values

timeouts

spider timeouts > grid timeouts > shuttle adapter timeouts

Logs

SW communication

spider grid shuttle

```
2021-06-25 02:16:02.437 INFO  [SHUT0057] Starting activity: { transporter: SHUT0057, MOVE_TO_LOC ( S03.019.0070.L ) }
2021-06-25 02:16:02.437 INFO  [SHUT0057_transport] moveToLocation({ transporter: SHUT0057, MOVE_TO_LOC ( S03.019.0070.L ) })
2021-06-25 02:16:02.437 INFO  [SHUT0057_transport] <-- drive(path: S03.019.0070.L, boxWidth: 400, checkRack: false)

2021-06-25 02:16:02.437 INFO  [SHUT0057] --> drive(path: [S03.019.03.01.0001.R, S03.019.03.01.0070], boxWidth: 400, checkRack: false)


2021-06-25 02:16:02.441 INFO  [SHUT0057_delegate] --> startDrive(orderId: 14431, boxWidth: 400, accelerationFactor: 1.0, positions: [S03.019.03.01.0001.R,
S03.019.03.01.0028])

2021-06-25 02:16:02.442 INFO  [SHUT0057] --> startDrive(orderId: 14431, targets: [(S03.019.03.01.0001.R, speed: 4000, target info [CHECKTAG]),
(S03.019.03.01.0028, speed: 4000, target info [CHECKTAG])], boxWidth: 400, accelerationFactor: 1.0)


2021-06-25 02:16:15.281 INFO  [SHUT0057] <-- finishedDrive(orderId: 14431, position: S03.019.03.01.0070, deviation: 1, channelSide: LEFT, boxWidth: 400,
errorInfo: [])

2021-06-25 02:16:15.282 INFO  [SHUT0057_delegate] <-- finishedDrive(position: S03.019.03.01.0070, deviation: 1, channelSide: LEFT, boxWidth: 400, errorInfo:
[])


2021-06-25 02:16:15.282 INFO  [SHUT0057] <-- drove(shuttleStatus: GridShuttleStatus(busy: false, reservations: [[S03.019.03.01.0070, S03.019.03.01.0070]],
positionStatus: (currentPositionId: S03.019.03.01.0070, lastPositionId: , deviation: 1, rfidTag: , boxWidth: 400, referenced: TTRUE, scaled: TTRUE,
orderActive: TFALSE), sensorStatus: (orientation: NORTH_SOUTH, isInCrossingArea: false, endzoneStatus: (isInEndzone: IS_IN_ENDZONE, endzoneDetail:
ENDZONE_NORTH), collDetLeft: true, collDetRight: true), inspectionState: IS_INACTIVE, failure: []), failure: [])

2021-06-25 02:16:15.283 INFO  [SHUT0057_transport] drove(status: { curPos: S03.019.03.01.0070, idle, referenced: TTRUE, scaled: TTRUE, orderActive: TFALSE,
boxWidth: 400, northEnd, inspection: IS_INACTIVE }, failure: None)
2021-06-25 02:16:15.283 INFO  [SHUT0057] Activity finished [  12846ms]: { transporter: SHUT0057, MOVE_TO_LOC ( S03.019.0070.L ) }
```

```
02.442 INFO  [SHUT0057] --> startDrive(orderId: 14431, targets: [(S03.019.03.01.0001.R, speed: 4000, target info [CHECKTAG]), (S03.019.03.01.0028, speed: 4000, target info [CHECKTAG])], boxWidth: 400, accelerationFactor:


02.442 DEBUG [SHUT0057] Sending Telegram: GetGeneralStatusTelegram: (version: 1, adr: 1, msgID: 3, options: 0, length: 3, FC: 10, SFC: 1, Ver: 1)
02.445 DEBUG [SHUT0057] Received telegram: GeneralStatusV3Telegram: (version: 1, adr: 1, msgID: 3, options: 1, length: 9, FC: 11, SFC: 1, Ver: 3)
                        (shuttle status: 0, position status: order act.: false, pos. act.: false, lift. act.: false, rack data: false, referenced: true, scaled: true, position sensor status: endz.: S|-,
                         crossing: false, orientation: N|S, blocked: -|-, lhdStatus: 32, error: 0)

02.445 DEBUG [SHUT0057] Sending Telegram: GetPositionStatusTelegram: (version: 1, adr: 1, msgID: 4, options: 0, length: 3, FC: 41, SFC: 1, Ver: 1)
02.448 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: 4, options: 1, length: 37, FC: 42, SFC: 1, Ver: 3)
                        (orderId: 340, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: -1461, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: false, pos. act.: false,
                        lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: S|-, crossing: false, orientation: N|S, blocked: -|-], error: 0|0, tag: 0000000000000000)

02.449 INFO  [SHUT0057] <-- startingToDrive(transporterId: 300000057, positions: [(S03.019.03.01, -1462mm), (S03.019.03.01, 12484mm)])

02.449 DEBUG [SHUT0057] Sending Telegram: MoveRackV2Telegram: (version: 1, adr: 1, msgID: 5, options: 0, length: 51, FC: 40, SFC: 20, Ver: 2) (orderId: 342)
                        (orderId: 342, boxWidth:400, boxWidthOffset0, acceleration: 1000, targets: 2 - [[(sector: 3.19.3.1-3.19.3.2, type: TRANSFER, mode: NORMAL, position: -1462, speed: 4000, side: false,
                        direction: false, target info: [CHECKTAG]), (sector: 3.19.3.1-3.19.3.2, type: IN_SECTOR, mode: NORMAL, position: 12484, speed: 4000, side: false, direction: false, target info: [CHECKTAG])]])
02.452 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: 5, options: 1, length: 37, FC: 42, SFC: 1, Ver: 3)
                        (orderId: 342, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: -1461, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: true, pos. act.: true,
                        lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: S|-, crossing: false, orientation: N|S, blocked: -|-], error: 0|0, tag: 0000000000000000)
02.452 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0001.R, deviation: 1.

02.496 INFO  [SHUT0057] --> updateDrive(orderId: 14431, targets: [(S03.019.03.01.0029, speed: 4000, target info [CHECKTAG]), (S03.019.03.01.0055, speed: 4000, target info [CHECKTAG])])
02.497 INFO  [SHUT0057] <-- updatingDrive(transporterId: 300000057, positions: [(S03.019.03.01, 12944mm), (S03.019.03.01, 26094mm)])

02.497 DEBUG [SHUT0057] Sending Telegram: MoveRackV2Telegram: (version: 1, adr: 1, msgID: 6, options: 0, length: 51, FC: 40, SFC: 20, Ver: 2) (orderId: 342)
                        (orderId: 342, boxWidth: 400, boxWidthOffset0, acceleration: 1000, targets: 2 - [[(sector: 3.19.3.1-3.19.3.2, type: IN_SECTOR, mode: NORMAL, position: 12944, speed: 4000, side: false,
                        direction: false, target info: [CHECKTAG]), (sector: 3.19.3.1-3.19.3.2, type: IN_SECTOR, mode: NORMAL, position: 26094, speed: 4000, side: false, direction: false, target info: [CHECKTAG])]])
02.660 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: 6, options: 1, length: 37, FC: 42, SFC: 1, Ver: 3)
                        (orderId: 342, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: -1461, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: true, pos. act.: true,
                        lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: S|-, crossing: false, orientation: N|S, blocked: -|-], error: 0|0, tag: 0000000000000000)

07.518 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: -60, options: 0, length: 37, FC: 42, SFC: 1, Ver: 3)
                        (orderId: 342, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: 12485, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: true, pos. act.: true,
                        lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: -|-, crossing: false, orientation: N|S, blocked: -|-], error: 0|0, tag: 0000000000000000)
07.518 DEBUG [SHUT0057] Sending response: AcknowledgeTelegram: (version: 1, adr: 1, msgID: -60, options: 1, length: 0)
07.518 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0028, deviation: 1.

.....................................................................................................................................................................................

15.281 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: -55, options: 0, length: 37, FC: 42, SFC: 1, Ver: 3)
                        (orderId: 342, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: 32295, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: false, pos. act.: false,
                        lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: -|N, crossing: false, orientation: N|S, blocked: L|R], error: 0|0, tag: 0000000000000000)
15.281 DEBUG [SHUT0057] Sending response: AcknowledgeTelegram: (version: 1, adr: 1, msgID: -55, options: 1, length: 0)
15.281 DEBUG [SHUT0057] Shuttle at position: S03.019.03.01.0070, deviation: 1.
15.281 INFO  [SHUT0057] <-- positionStatusSupervisionAvailable(transporterId: 300000057, position: (S03.019.03.01, 32295mm), rfidTag: 0000000000000000, referenced: TTRUE, standing: TTRUE)

15.281 INFO  [SHUT0057] <-- finishedDrive(orderId: 14431, position: S03.019.03.01.0070, deviation: 1, channelSide: LEFT, boxWidth: 400, errorInfo: [])
```

2021-06-25 02:16:15.287 INFO [SHUT0057] --> **storeTrays**(locationId: S03.019.0070.L, containersToStore: [CID: 200011668, maxPos: 1, rp: 1, type: 10], containerBehind: [CID: 200049480, maxPos: 2, rp: 1, type: 10], allowTelescopeExtensionToOppositeSide: false, correctPositionOfContainerBehind: false, limitTelescopeExtension: false, speedFactor: 1.0)

2021-06-25 02:16:15.287 DEBUG [SHUT0057] Running operation: **StoreMultipleJobsOperationManager**(id: **4378612**, state: [NEW], failure: None)
2021-06-25 02:16:15.287 DEBUG [SHUT0057] Starting StoreMultipleJobsOperationManager(id: 4378612, state: [NEW], failure: None)
2021-06-25 02:16:15.287 DEBUG [SHUT0057] Sending Telegram: GetPositionStatusTelegram: (version: 1, adr: 1, msgID: 8, options: 0, length: 3, FC: 41, SFC: 1, Ver: 1)
2021-06-25 02:16:15.297 DEBUG [SHUT0057] Received telegram: PositionStatusV3Telegram: (version: 1, adr: 1, msgID: 8, options: 1, length: 37, FC: 42, SFC: 1, Ver: 3) (orderId: 342, positioning mode: 20, target mode: 0, sector: S03.019.03.01, position: 32295, boxWidth: 400, boxWidthOffset: 0, position status: [order act.: false, pos. act.: false, lift. act.: false, rack data: false, referenced: true, scaled: true], sensor status: [endz.: -|N, crossing: false, orientation: N|S, blocked: -|-], error: 0|0, tag: 0000000000000000)
2021-06-25 02:16:15.297 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [RUNNING], failure: None) -- start processing telegram
2021-06-25 02:16:15.297 DEBUG [SHUT0057] Sending Telegram: **StoreJobsTelegram**: (version: 1, adr: 1, msgID: 9, options: 0, length: 35, FC: 50, SFC: 16, Ver: 3) (orderId: 343, location type: 2, location depth: 1325, location mode: 20, location fill: 688, side: 0, speed: 700, jobs: 1, trays: [(type: 5, length: 600, position: 48, tray offset: 22, pos. offset: 0, handling retries: 25x2mm)]))
2021-06-25 02:16:15.301 DEBUG [SHUT0057] Received telegram: **LoadStatusV2Telegram**: (version: 1, adr: 1, msgID: 9, options: 1, length: 9, FC: 52, SFC: 1, Ver: 2) (orderId: 343, status: 33, mode: 16, error: 0|0)
2021-06-25 02:16:15.301 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [RUNNING], failure: None) -- start processing telegram
2021-06-25 02:16:15.302 DEBUG [SHUT0057] Stopping position timeout because operation finished.

2021-06-25 02:16:23.602 DEBUG [SHUT0057] Received telegram: **LoadStatusV2Telegram**: (version: 1, adr: 1, msgID: -53, options: 0, length: 9, FC: 52, SFC: 1, Ver: 2) (orderId: 343, status: 0, mode: 16, error: 0|0)
2021-06-25 02:16:23.602 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [RUNNING], failure: None) -- start processing telegram
2021-06-25 02:16:23.602 DEBUG [SHUT0057] Sending response: AcknowledgeTelegram: (version: 1, adr: 1, msgID: -53, options: 1, length: 0)
2021-06-25 02:16:23.602 DEBUG [SHUT0057] Sending Telegram: **GetGeneralStatusTelegram**: (version: 1, adr: 1, msgID: 10, options: 0, length: 3, FC: 10, SFC: 1, Ver: 1)
2021-06-25 02:16:23.603 DEBUG [SHUT0057] Received telegram: PowerStatusTelegram: (version: 1, adr: 1, msgID: -52, options: 0, length: 20, FC: 26, SFC: 1, Ver: 1) (type: POWER_OK, input current: 1040mA, limit: 10000mA, input volt: 47649mV, buffer volt: 57519mV)
2021-06-25 02:16:23.603 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [RUNNING], failure: None) -- start processing telegram
2021-06-25 02:16:23.624 DEBUG [SHUT0057] Received telegram: **GeneralStatusV3Telegram**: (version: 1, adr: 1, msgID: 10, options: 1, length: 9, FC: 11, SFC: 1, Ver: 3) (shuttle status: 0, position status: order act.: false, pos. act.: false, lift. act.: false, rack data: false, referenced: true, scaled: true, position sensor status: endz.: -|N, crossing: false, orientation: N|S, blocked: -|-, lhdStatus: 0, error: 0)
2021-06-25 02:16:23.624 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [RUNNING], failure: None) -- start processing telegram
2021-06-25 02:16:23.624 DEBUG [SHUT0057] StoreMultipleJobsOperationManager(id: 4378612, state: [FINISHED], failure: None) Operation finished.
2021-06-25 02:16:23.624 INFO [SHUT0057] <-- **storedTrays**(locationId: S03.019.0070.L, storedContainers: [{ id: 200011668, pos: 1, rp: 1 }], errorInfo: [])

Questions?

Thank you.

Vielen Dank.

Merci beaucoup.

Muchas gracias.

Mille grazie.

KNAPP

PLC Process

Handout

TODO:

TODO:

08/2018

Table of contents

1. Standard-Configuration

The plc process is used for the communication with the plc. There should be one plc-process for each CPU in the system. Normally the processes are named according to the naming of the CPUs (e.g. plca for CPU a).

The file in which the plc-stations are configured is the plc_adapter_cfg.py. In this file all the stations from the plc are configured, e.g.:

```
station1140 = OSR.PlcAdapterCfg.StationCfg(
    stationId = 1140,
    type = OSR.PlcAdapterCfg.SimpleStation,
    containerFormat = "\0", # Use "T" for OSR15 trays, "\0" otherwise
    destMap = [],
    description = "Loop switch",
    group = "")
```

The station-ID must fit the station-ID used by the plc (they normally can be found in the station-communication file (sometimes there might be differences – to be discussed with the plc-startup person)). Then the right station type needs to be configured. All different station types are described in the idl-file. An appropriate description for each station should be found. This description is then shown in the Station-Info in the reports. Also, the shuttle stations need to be configured:

```
shuttleStations = [
    OSR.PlcAdapterCfg.StationCfg(
    stationId = int((a * 1000 + l * 10 + 1) - 0x080000000),
    type = OSR.PlcAdapterCfg.ShuttleStation,
    containerFormat = "\0",                                    # unused
    destMap = [],                                              # unused
    description = "Shuttle in aisle %s level %s" % (a, l),
    group = "")
    for (a, l) in shuttlesInAisles]
```

Then the stations need to be added to the station list.

```
# Now add the shuttle stations to the station list.
                                #min/max aisle, min/max level
stations.extend(getShuttlesInAisles(1,4, 1,22))
```

For each CPU an array with the controlled stations must be created:

```
stationIdsA =   [11900, 11600, ... ]
```

The shuttle-stations need to be added to the right array as well.

This array and further parameters should be configured for each CPU:

```
plc_configA = OSR.PlcAdapterCfg.Config(
   stations = stationIdsA,
   plcIpAddress = "osrplcA",
   localTSAP = "\xE0\x02\x33",
   remoteTSAP = "\xE0\x02\x33",
   tlvCatLogLevel = OSR.LL_INFO,
   maxPacketsPerTSDU = 5,
   maxBytesPerTSDU = 500,
   maxBytesPerTPDU = 0,
   reconnectInterval = 5000,
   acknowledgementTimeout = 2000,
   heartBeatInterval = 10000,
   plcPort = 102)
```

As plcIpAddress the hostname, which was inserted in /etc/hosts should be used.

At the end of the file the plca_adapter and the array with stations should be handed over:

```
try:
    osr_ctx.unbind(["plc_adapter1"])
except:
    pass

osr_ctx.bind(["plc_adapter1"],
             CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/Config:1.0"),
                       plc_config))


try:
    osr_ctx.unbind(["station_cfg_list"])
except:
    pass

osr_ctx.bind(["station_cfg_list"],

CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/StationCfgList:1.0"),
                       station_cfg_list))
```

The new executed config stuff should be used for historizing the executed cfgs.

```
#--> new current executed cfg stuff
import executed_config
executed_config.historizing(osr_id,"plcb_cfg")
```

Another configuration file, which is indirectly important for the plc-process (or to be more accurate for the right movement of the lifts over the wsc) is the hmp_cfg.py. There the lift stations, their entrance readings and baselevels need to be defined:

```
stations = {
# (aisle_id, lift_id):(lift_station, entrances (station_id, level), exit_levels,
lift_type,          lift_name, is_aisle_lift)
  (1,        1):      (11600,        [(12900, 101)],                [],
HMPCfg.QuadLift,    None,      True),
  (1,        2):      (21600,        [],                [201],
HMPCfg.QuadLift,    None,      True),
  }
```

2. Process Start

For starting up of the plc processes a run-file and a log-run-file are needed. Therefore the templates can be inserted into the plc-folder.

```
mkdir plc
cd plc
m4 --define=M4_OSR_ID=osr1 $OSR_HOME/templates/plc.run_SuSE > run
chmod 0774 run
mkdir log
cd log
m4 --define=M4_OSR_ID=osr1 $OSR_HOME/templates/plc-log.run_SuSE > run
chmod 0774 run
```

In the run file an if-query can be found – to be sure, that gcs and naming are running, before the plc-process starts.

```
if ! isSvcUp naming 2 || ! isSvcUp gcs 10 ; then
    sleep 1
    exit 1
fi
```

When the executed-config-stuff is used, also this line needs to be in the run-file:

```
# cleanup configs
python /home/osr/knapp/lager/lib/python/executed_config.py --osr_id $OSR_ID
--processname "plc_cfg" --cleanup
```

Also important is, that in the following line the right plc_adapter is inserted for each plc-process:

```
at.syslog.osr.adapters.plc.Main $OSR_ID plc_adapter1 true
```

Depending if the new GzipRoller is used, the log-runfile will look a little bit different. The new version (with the new roller) should look like that:

```
#!/bin/bash

umask 0133
exec $OSR_HOME/bin/roller -b $OSR_HOME/inst/osr1/log/plc -c
$OSR_HOME/inst/osr1/cfg/roller.cfg
```

At the end for starting the process a link in the service-folder needs to be created:

```
cd $OSR_HOME/inst/osr1/service
ln -s $OSR_HOME/inst/osr1/_service/plc .
```

After startup of the plc-process, when there are containers driving (and the steuer process is up and running) the log-messages for loop stations will look similar to the following:

```
[S12500] <-- containerClamped(CID: 111670, UNK: 1, RE: 0, HF: 0, SF: 0)
[S12500] --> announceContainer(CID: 111670, SRC: 0, DEST: -1, HEIGHT: 1)
[S12500] <-- containerProcessed(CID: 111670, POS: -1, UNK: 0, RE: 0, HF: 0, SF:
0, OOW: 0, OH: 0, OC: 0, NB: 0, TU: 0)
[S12500] --> deleteContainer(CID: 111670)
```

When the container arrives at the station, the plc sends us a containerClamped. Then the steuer sends a announceContainer with the wanted destination (most often 0 for staying on the loop and -1 for diverting at the station)
Then we get a containerProcessed message with the position, where the plc sent the container. A lot of information on the processing is included. The most common abbreviations stand for:

UNK	unknown container
RE	reading error
HF	hardware failure
SF	station full
OOW	out of window; motor moving too slowly
OH	over-height
	NB – no box

Thereafter you will find a deleteContainer message, so that the plc station should forget the container.

For a lift-station you might see something like that:

```
[S21600] --> moveToLevel(LEV: 201, LHD: 1)
[S21600] <-- movedToLevel(LEV: 201, LHD: 1, ULvl: 0, ULhd: 0, HF: 0, EC:
0x01000000)

[S21600] --> checkLocationWithLhd(LHD: 1)
[S21600] <-- checkedLocation(LVL: 201, occupied: false, notAtLevel: false,
unknownLevel: false, hardwareFailure: false, EC:
 0x07000000)

[S21600] --> checkLhd(LHD: 2)
[S21600] <-- checkedLhd(LHD: 2, LhdOcc: 0, ULhd: 0, HF: 0, EC: 0x07000000)

[S21600] --> load(LHD: 1)
[S21600] <-- loaded(LEV: 10, LHD: 1, NB: 0, CID: 0, RE: 0, LhdOcc: 0, ULhd: 0,
NAL: 0, HF: 0, EC: 0x02000000)

[S21600] --> store(LHD: 2, measureBoxWidth: 0)
[S21600] <-- stored(LEV: 201, LHD: 2, NB: 0, TgtOcc: 0, ULhd: 0, NAL: 0,
NoCarrier: 0, HF: 0, EC: 0x03000000)
```

The lift gets different orders (normally from spider) – like move, store, load and so on. Then there will always be the feedback of the order sent by the lift (also including possible error information).

3. Good Practice

To have a better overview, a helper-function can be defined for the definition of the stations.

```
def getStation(stationId, type, containerFormat, destMap,
description, group):
    return OSR.PlcAdapterCfg.StationCfg(
        stationId = stationId,
        type = type,
        containerFormat = containerFormat,
        destMap = destMap,
        description = description,
        group = group)
```

Then the station-definitions can be done “inline”.

```
getStation(11600, OSR.PlcAdapterCfg.QuadLift, "\0", [], "Aisle 1 Lift 1", "Lifts")
```

Further the order of the definitions should be comprehensible. Grouping of the stations should be preferred (e.g. all pickstation-readings after each other).

The configuration of the stations for the different CPUs can also be done in this way:

```
"osrplcB": [
            getStation(11600, OSR.PlcAdapterCfg.QuadLift, "\0", [], "Aisle 1 Lift
1", "Lifts"),
            getStation(21600, OSR.PlcAdapterCfg.QuadLift, "\0", [], "Aisle 1 Lift
2", "Lifts"),
           ]
```

Further also the shuttle-stations can be configured and added to the right CPU

```
for aisle in range(1, aisleCount + 1):
    for level in range(1, levelCount +1):
        stations["osrplcB"].append(getStation(int((aisle * 1000 + level * 10 + 1)
- 0x080000000),
                                                OSR.PlcAdapterCfg.ShuttleStation,
"\0", [], "SHUT0%d%02d1" % (aisle, level), ""))
```

Then the plc-configs can be added like this:

```
#--> the plc configs
plcs = {}
plcs["osrplcB"] = getConfig("osrplcB")
plcs["osrplcC"] = getConfig("osrplcC")
```

Using the helper-method:

```
def getConfig(plc):
    return OSR.PlcAdapterCfg.Config(
        stations = [x.stationId for x in stations[plc]],
        plcIpAddress = plc,
        localTSAP = "\xE0\x02\x33",
        remoteTSAP = "\xE0\x02\x33",
        tlvCatLogLevel = OSR.LL_INFO,
        maxPacketsPerTSDU = 5,
        maxBytesPerTSDU = 1000,
        maxBytesPerTPDU = 0,
        reconnectInterval = 5000,
        acknowledgementTimeout = 2000,
        heartBeatInterval = 10000,
        plcPort = 102)
```

Then the plc_adapters can be handed over like this:

```
for plc in plcs:
    try:
        osr_ctx.unbind(["plc_adapter%s" % plc])
    except:
        pass
    osr_ctx.bind(["plc_adapter%s" % plc],

CORBA.Any(CORBA.TypeCode("IDL:OSR/PlcAdapterCfg/Config:1.0"),
                            plcs[plc]))
```

4. Common Errors

Some common errors would be, that the StationID was configured different to the ID used by the plc. Further there could also be a error, if the wrong station type is used in the configuration. Another error could be, that the station was added to the wrong CPU. When testing with the simulator is done, it must be checked if the CPU-configuration of the simulator matches the one of the plc-configuration.

Status	Finished
Started	Wednesday, 25 June 2025, 2:28 PM
Completed	Wednesday, 25 June 2025, 2:33 PM
Duration	4 mins 46 secs
Marks	8.36/10.00
Grade	**83.57** out of 100.00

Question 1

Partially correct

Mark 0.50 out of 1.00

Which of the following statements concerning adapter processes are correct? Check all that apply:

- a. Every adapter process needs access to the wh_model.xml file
- b. Adapter processes do not connect to the OSR database
- c. Adapter processes provide an interface between main processes and some hardware devices
- d. The adapter processes are the last ones to start during SRC startup

Your answer is partially correct.

You have correctly selected 1.

The correct answers are:
Adapter processes provide an interface between main processes and some hardware devices,

Adapter processes do not connect to the OSR database

Question 2

Correct

Mark 1.00 out of 1.00

Which of the statements below are correct for shuttle related processes? Check all that apply:

- [x] a. The shuttle adapter process needs to read the wh_model.xml
- [x] b. The shuttle process must be able to reach the shuttles on network level
- [x] c. For Evo systems, the grid adapter is between spider and shuttle
- [ ] d. Routing information is configured in the grid_cfg.py
- [ ] e. Each OSR level has its own shuttle configuration file
- [x] f. In a simulator environment, it is ok to remove the "!TODO!" entries from the shuttle config
- [x] g. Most of the configurations in the shuttle config are entered by shuttle commissioning
- [ ] h. Shuttle configuration is not stored within gcs, but in shuttle_cfg.xml

Your answer is correct.

The correct answers are:
The shuttle process must be able to reach the shuttles on network level,

For Evo systems, the grid adapter is between spider and shuttle,

Most of the configurations in the shuttle config are entered by shuttle commissioning,

The shuttle adapter process needs to read the wh_model.xml,

In a simulator environment, it is ok to remove the "!TODO!" entries from the shuttle config

Question 3

Correct

Mark 1.00 out of 1.00

In an **Evo** system, there is a direct communication between spider and shuttle.

- ( ) True
- (•) False

The correct answer is 'False'.

Question 4

Correct

Mark 1.00 out of 1.00

Which process will only exist in an Evo 2D system?

a. shuttle

b. two_d

c. grid

d. cruise_control ✓

e. evo2_shuttle

Your answer is correct.

The correct answer is:
cruise_control

Question 5

Correct

Mark 1.00 out of 1.00

In what sequence do messages appear in a PLC log file for a station in the prezone?

- ✓ containerClamped
- ✓ announceContainer
- ✓ containerProcessed
- ✓ deleteContainer

Your answer is correct.

Question 6

Correct

Mark 1.00 out of 1.00

When you are unsure if the `RackStation` from the example below is the correct station type -- or generally if you want to know which station types for PLC stations are valid -- in which file could you look this up?

```
##### Rack Admin Station #####
# Depending on the inspection set-up of the PLC, only one or one per
# CPU is needed. If more than one is needed, the ids have to be
# different.
# If configured, then shuttle stations are not necessary.
createPlcStation(9999, OSR.PlcAdapterCfg.RackStation, "Logging Stations from SPS", "MESSAGE-STATION"),
```

Answer: plc_adapter_cfg.idl

The correct answer is: plc_adapter_cfg.idl

Question 7

Correct

Mark 1.00 out of 1.00

Match the following arguments in the responses from PLC stations to their meanings:

UNK	unknown container
SF	station full
OH	over-height
HF	hardware failure
OOW	out of window (motor moving too slowly)
RE	reading error

Your answer is correct.

The correct answer is:
UNK → unknown container,

SF → station full,

OH → over-height,

HF → hardware failure,

OOW → out of window (motor moving too slowly),

RE → reading error

Question 8

Not answered

Marked out of 1.00

What does a "**1**" as the *__second__* digit in the number of a prezone PLC station tell you?

a. This is a lift station

b. This station belongs to OSR1

c. This is an entrance to an aisle

d. This station is next to a picking station

e. This is the lower conveyor line, so it will be for targets

Your answer is incorrect.

The correct answer is:
This is the lower conveyor line, so it will be for targets

Question 9

Partially correct

Mark 0.86 out of 1.00

Following the standard nomenclature for PLC station numbers, match the stations below to their corresponding types:

12502	Choose...
12403	prezone helper station ✓
12900	lift entrance reading ✓
12802	picking station ✓
21600	OSR lift ✓
11704	diversion to picking station ✓
11601	OSR lift ✓

Your answer is partially correct.

You have correctly selected 6.

The correct answer is:
12502 → diversion to an aisle,

12403 → prezone helper station,

12900 → lift entrance reading,

12802 → picking station,

21600 → OSR lift,

11704 → diversion to picking station,

11601 → OSR lift

Question 10

Correct

Mark 1.00 out of 1.00

How many plc processes exist in an OSR instance?

a. As many processes as there are PLC CPUs ✓

b. One maximum

c. One process per level of the warehouse building

d. One per connected PLC station

e. Two maximum

Your answer is correct.

The correct answer is:
As many processes as there are PLC CPUs

Status	Finished
Started	Wednesday, 25 June 2025, 2:28 PM
Completed	Wednesday, 25 June 2025, 2:33 PM
Duration	4 mins 46 secs
Marks	8.36/10.00
Grade	**83.57** out of 100.00

Question 1

Partially correct

Mark 0.50 out of 1.00

Which of the following statements concerning adapter processes are correct? Check all that apply:

- a. Every adapter process needs access to the wh_model.xml file
- b. Adapter processes do not connect to the OSR database
- c. Adapter processes provide an interface between main processes and some hardware devices
- d. The adapter processes are the last ones to start during SRC startup

Your answer is partially correct.

You have correctly selected 1.

The correct answers are:
Adapter processes provide an interface between main processes and some hardware devices,

Adapter processes do not connect to the OSR database

Question 2

Correct

Mark 1.00 out of 1.00

Which of the statements below are correct for shuttle related processes? Check all that apply:

- [x] a. The shuttle adapter process needs to read the wh_model.xml
- [x] b. The shuttle process must be able to reach the shuttles on network level
- [x] c. For Evo systems, the grid adapter is between spider and shuttle
- [ ] d. Routing information is configured in the grid_cfg.py
- [ ] e. Each OSR level has its own shuttle configuration file
- [x] f. In a simulator environment, it is ok to remove the "!TODO!" entries from the shuttle config
- [x] g. Most of the configurations in the shuttle config are entered by shuttle commissioning
- [ ] h. Shuttle configuration is not stored within gcs, but in shuttle_cfg.xml

Your answer is correct.

The correct answers are:
The shuttle process must be able to reach the shuttles on network level,

For Evo systems, the grid adapter is between spider and shuttle,

Most of the configurations in the shuttle config are entered by shuttle commissioning,

The shuttle adapter process needs to read the wh_model.xml,

In a simulator environment, it is ok to remove the "!TODO!" entries from the shuttle config

Question 3

Correct

Mark 1.00 out of 1.00

In an **Evo** system, there is a direct communication between spider and shuttle.

- ( ) True
- (•) False

The correct answer is 'False'.

Question 4

Correct

Mark 1.00 out of 1.00

Which process will only exist in an Evo 2D system?

a. shuttle

b. two_d

c. grid

d. cruise_control ✓

e. evo2_shuttle

Your answer is correct.

The correct answer is:
cruise_control

Question 5

Correct

Mark 1.00 out of 1.00

In what sequence do messages appear in a PLC log file for a station in the prezone?

- ✓ containerClamped
- ✓ announceContainer
- ✓ containerProcessed
- ✓ deleteContainer

Your answer is correct.

Question 6

Correct

Mark 1.00 out of 1.00

When you are unsure if the `RackStation` from the example below is the correct station type -- or generally if you want to know which station types for PLC stations are valid -- in which file could you look this up?

```
##### Rack Admin Station #####
# Depending on the inspection set-up of the PLC, only one or one per
# CPU is needed. If more than one is needed, the ids have to be
# different.
# If configured, then shuttle stations are not necessary.
createPlcStation(9999, OSR.PlcAdapterCfg.RackStation, "Logging Stations from SPS", "MESSAGE-STATION"),
```

Answer: plc_adapter_cfg.idl

The correct answer is: plc_adapter_cfg.idl

Question 7

Correct

Mark 1.00 out of 1.00

Match the following arguments in the responses from PLC stations to their meanings:

UNK	unknown container
SF	station full
OH	over-height
HF	hardware failure
OOW	out of window (motor moving too slowly)
RE	reading error

Your answer is correct.

The correct answer is:
UNK → unknown container,

SF → station full,

OH → over-height,

HF → hardware failure,

OOW → out of window (motor moving too slowly),

RE → reading error

Question 8

Not answered

Marked out of 1.00

What does a "**1**" as the *__second__* digit in the number of a prezone PLC station tell you?

- a. This is a lift station
- b. This station belongs to OSR1
- c. This is an entrance to an aisle
- d. This station is next to a picking station
- e. This is the lower conveyor line, so it will be for targets

Your answer is incorrect.

The correct answer is:
This is the lower conveyor line, so it will be for targets

Question 9

Partially correct

Mark 0.86 out of 1.00

Following the standard nomenclature for PLC station numbers, match the stations below to their corresponding types:

12502	Choose...
12403	prezone helper station ✓
12900	lift entrance reading ✓
12802	picking station ✓
21600	OSR lift ✓
11704	diversion to picking station ✓
11601	OSR lift ✓

Your answer is partially correct.

You have correctly selected 6.

The correct answer is:
12502 → diversion to an aisle,

12403 → prezone helper station,

12900 → lift entrance reading,

12802 → picking station,

21600 → OSR lift,

11704 → diversion to picking station,

11601 → OSR lift

Question 10

Correct

Mark 1.00 out of 1.00

How many plc processes exist in an OSR instance?

a. As many processes as there are PLC CPUs ✓

b. One maximum

c. One process per level of the warehouse building

d. One per connected PLC station

e. Two maximum

Your answer is correct.

The correct answer is:
As many processes as there are PLC CPUs