steuer

General overview

steuer is one of the three main processes – and as such it is responsible for controlling the prezone. Here, steuer's main task is to ensure proper routing of source and target containers to their designated destinations. In order to do so, steuer needs the following information:

- target(s) for a container
- its current position
- a "routing table", explaining how to reach one place from another

When SRC receives a pick order and a target container from the host system, steuer sends the target container to a picking station and requests suitable source containers from the OSR.

Configuration

In the steuer config, first all OSR aisles (*"towers"*), PLC and picking stations need to be defined. For the PLC stations, a distinction must be made between *loop switches* (stations on the source loop) and *diverters* (stations on the target line). Also, some stations need to be defined specifically – e.g. *fill readers* which are PLC stations on the source loop where a decision must be made for a container on which aisle will be chosen. In the definitions for the picking stations, there are settings (***fetch*** and ***prefetch***) which control how many containers will be requested from the OSR at the same time (per picking station).

In the ***layout*** definition, the structure of the pool(s) will be done – if there is more than one conveyor system, an entry must be made for each of those systems (they must correspond to the "***pools***" definition in the warehouse model!)

The ***loopLayout*** definition holds the routing information; it explains for each station which destinations can be reached, depending on what direction is chosen.

steuer

Christian Bretterklieber

v 1.1

Table of Contents

Overview

steuer is one of the three main processes within SRC each of which is handling a specific area, and as such it is responsible for controlling the prezone – the area which connects the OSR and the picking stations with its conveyor systems.

Functions

The steuer process controls the flow of containers through the conveyor system, and is responsible that containers arrive at the correct stations in a timely manner. In general, steuer's tasks are:

- overall control of the prezone
- storing containers to the OSR
- handling pick orders
- coordination between stations
  - prezone (PLC) stations
  - picking stations
  - OSR aisles

Connections

As steuer is coordinating between all the stations, it has a lot of communication interfaces to communicate with any involved parties; hence it communicates with:

- **plc adaptor** to receive information about containers that are clamped at PLC stations, and to (pre-) announce them to stations
- **spider** to request containers and announce when a container will be sent to the OSR
- **opa** to inform the picking stations about containers they are receiving for a pick order
- the **database** to read any required information about containers

General overview and important terms

Source and target containers

In the case where the SRC system is used for handling warehouse picking, we need to differentiate between containers that are used for storing articles in the OSR (and subsequently for taking goods ***out of them*** during order picking), and containers that are designated to hold goods picked ***into them***.

While there are several terms for containers (like "*tray*", "*tote*", "*box*", and so on), and they are sometimes used interchangeably, it is good to note that when we want to differentiate, we use "***trays***" when we are talking about the sources, and "***containers***" when we talk about the target totes.

Source trays

Source trays are containers that are generally stored in the OSR, and where the warehouse pickers will take the articles out of. As for these containers, the SRC system needs to have a lot of information tss it have compartments, which – and how many – articles are stored, where is it located, etc.), all this information gets stored in the database he main table is **CONTAINERS** here and every tray has a record; then there are several tables which hold related data, like **CONTAINER_LOCATIONS** (where is a container currently located), **SLOTS** (compartments in the containers), **SLOT_CONTENTS** (products stored in the compartments), etc.

In order to comfortably store trays in the OSR, very often standardized plastic containers are used, but it's also possible that customers use cardboard boxes for storage.

From the steuer perspective, a source tray always has tasks associated to it (like going to one or more picking stations); steuer maintains this task list, and bases the container's routing (see below) on those tasks.

Target containers

Target containers are the totes into which products are placed during the picking process. They are as such not stored in the **CONTAINERS** table of the OSR database.

In general, there are several possibilities as to how and where a target container can come into the system – for one, WCS can send target containers in through the conveyor system (in that case, the containers are already "***married***" to an order); otherwise, target containers can be manually placed (and at that time, assigned to an order) at a picking station when the SRC orderstart is used.

Conveyor systems

Conveyors are the main means of transportation within the prezone. Technically, there are several types of conveyors – using either carrier rollers or carrier belts. Along the conveyor system, there will be many PLC stations with which steuer will communicate. Very often, these PLC stations are at junctions, and there they control whether a container stays on the conveyor system, or whether it will leave the main conveyor ("***divert***").

When communicating with the PLC stations, steuer uses numeric codes to designate the direction a container needs to go:

- "**0**" stands for staying on the loop, "keep going"
- "**-1**" means "divert" and leave the conveyor. In most cases, a junction only allows to divert to one other route; whether this exit is to the left or the right side of the conveyor, does not matter
- in case there is a second diversion route possible at a junction, in addition to the direction "**-1**" you will also have a direction "**1**" – which one will then lead where (e.g. "-1" for right and "1" for left) needs to be verified with the PLC commissioner.

Source conveyor loop

The source conveyor loop is the conveyor system that connects the OSR with the picking stations. As this is a loop, containers moving along the conveyor system will stay indefinitely on it as long as they are not ordered to divert. Therefore, if a source container for instance passes its designated picking station (maybe it cannot divert as the target is filled up), it will eventually return there after it took another round along the complete loop.

Yet, as source loops can be quite long, and therefore the time for a container to travel a full round can be lengthy, such situations should be avoided as much as possible.

Target conveyor

The conveyor system for the target containers is usually not constructed as a loop; rather it is a one-directional conveyor which allows for sending the boxes out to the WCS area. It is possible that the target conveyor line starts outside somewhere in the WCS area, and containers are sent in from WCS, but it could also be that there is no incoming line but target containers are placed on the target conveyor directly at the SRC picking stations.

PLC stations

PLC stations are installed within the SRC area to control the physical flow of containers. Within the steuer controlled area, we will find PLC stations both on the source loop and on the target line. From a technical perspective, source loop stations and target line stations are the same, yet from the software point of view, they are implemented differently.

Loop switch

Loop switches are the PLC stations that are installed on the source conveyor loop. In the steuer configuration, they need to be entered in the `loopSwitchSequence` list.

Diverter

Diverters are PLC stations installed on the target conveyor system. Historically, there used to be several different class implementations on the software side, depending on the functionality required ("*toggling diverters*", "*splitting diverters*" and so on), nowadays, the practice is to use just the "***generic diverter***" class which comprises all functionalities.

In the configuration, all stations need to end up then in the `genericDiverterSequence` list.

Special handling for stations

From a functionality point of view, some stations are treated in a special way by steuer. Therefore, the following terms are introduced – and specific configurations are required in steuer:

Fill reader

A fill reader is from a technical point of view essentially a loop switch like any other – the difference is that from the software perspective, it is also viewed as a decision station: at that point the decision must be taken to which aisle of the OSR a container will be sent. Usually, the last PLC station on the source loop before the OSR aisles start (when following the source loop) is designated as a fill reader – depending on the prezone layout, there can be more than one fill reader.

Fetch station

A fetch station is the place where a container is considered to be fetched from the OSR (this is relevant for steuer's fetching algorithms). This could be a designated PLC station, or even just an OSR aisle exit.

Prezone structure

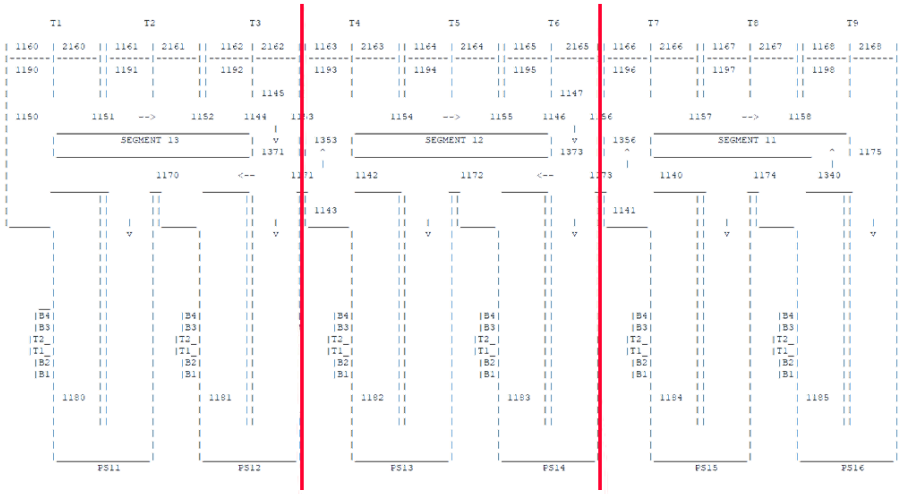
Floors

Out of various reasons (either the physical structure of the building requires it, or we need to improve the performance, as there are limits concerning the possible throughput on a conveyor), there is the possibility that there is more than one conveyor loop connecting OSR and picking stations. Each of these conveyor loops is referred to as a "***floor***" for steuer, and needs to have its own configuration.

Segments

In an unsegmented layout, all picking stations can be reached from all aisles of the OSR. However, in order to reduce travel time for containers on the loop, it is possible to introduce different segments in the layout – this requires additional diversion possibilities and PLC stations along the conveyor system. In a segmented system, while there is still a physical conveyor loop covering all the stations, the picking stations are logically assigned only specific OSR aisles, e.g. in the example below, the picking stations PS11 and PS12 are supplied only by aisles 1 – 3.

The disadvantage of segmentation is that the warehouse operator would have to maintain the same articles across all segments – otherwise, target containers may have to visit several picking stations in order to have their assigned orders completely fulfilled.



Prezone segmentation

Entrances and exits

As SRC is not an encapsulated system, but rather has connections to the “outside world” which is the WCS area, we also have to introduce the system boundaries to the steuer layout configuration, so we usually have at least one:

- **Entrance**: This is the point where containers enter the SRC system, and SRC takes over the responsibility. We denote an entrance with the virtual station number “***90***”.
- **Exit**: Here the containers leave the SRC realm, and responsibility goes to WCS. The primary exit gets the station number “***91***” assigned. There can be several physical exits (e.g. one exit for empty containers, another for transport orders, and so on). In that case, we would use sequential numbering – *91*, *92*, etc.

The entrance and exit stations are the boundaries where SRC interfaces with the host system; usually when containers pass by here, there should be a message sent by the *HIO* interface.

PLC communication

Generally, in the communication flow between plc and steuer, there are the following messages exchanged for regular PLC stations:

- "***containerClamped***": this is an incoming message, sent by PLC, which informs steuer that a container has arrived at a PLC station (the container's barcode has been read by the station), and PLC is asking SRC for further instructions.
- "***announceContainer***": this is an outgoing message that steuer sends to plc where it informs the process what to do with the container (i.e. whether to divert or stay on the loop).
- "***containerProcessed***": this is again an incoming message from PLC which informs SRC what it has done with the container. Please note that the direction that the PLC station has sent the container may deviate from steuer's order (e.g. in a case where the indicated conveyor is full, or when there is a physical failure).
- "***deleteContainer***": the final message that SRC sends to PLC to delete the information about the container from its ring buffer.

Pre-announcement

Through the steuer configuration, it is possible to define that from one station to another, the arrival of a container can be pre-announced. So when a container has been processed by one PLC station, steuer will already instruct the subsequent station that a container will arrive, and what to do with it.

This will speed up the container flow – in the communication exchange, this will have the effect that there is no more "***containerClamped***" message sent as the PLC station already knows what to do with the container and does not have to query SRC.

Routing in steuer

As steuer is responsible for controlling the flow of containers along the conveyors, it must know about all possible routes as well as targets that a container needs to reach.

Routing information

For every junction, steuer needs to know which targets can be reached from here by choosing any direction. This routing table information is stored in steuer config's `loopLayout` list.

Container targets

When a source container is under steuer's control, steuer maintains a list of all the tasks that are associated to the container – e.g. which picking station(s) it has to go to.

Handling storage to the OSR

Containers that need to go into the OSR (either containers that come from WCS via a "*goods-in*" process, or containers that were at a picking station and are returning now) are announced by steuer to spider.

The fill reader is the decision station where steuer will choose the aisle it will send the container to. It is possible to have steuer decide the designated aisle, and it will take the fill grade of the respective tower into account when doing so. It is also possible to let spider take the decision – in that case the steuer config parameter `storageSystemTransporterId` needs to be set to "**1**".

Workflow for selecting the aisle

- WCS sends a (e.g. transport) order to SRC; this message contains container information
- spider gets a "**CC**" ("*container class*") message via AQ and assigns a storage zone for that container in the database (based on the knowledge it has about the container and its class)
- At the fill reader, steuer picks an aisle which is assigned to the correct target zone (or has spider do so)
- steuer announces the container to spider

Handling pick orders

It is steuer's task to also coordinate pick orders; therefore, steuer will:

- receive and start pick orders
- send target containers to the picking stations (if target containers come from WCS)
- request the corresponding source trays from the OSR
- route source trays to the assigned picking stations

There are generally 2 possibilities for starting an order: either WCS is responsible, or – if WCS does not send target containers – SRC can handle the order start.

Orders with WCS connection

If WCS sends target containers to the SRC system, the handling of the orders in sequence is determined by the host system. The workflow is then as follows:

- WCS announces a pick order via the HIO interface
- WCS sends the target container via the target conveyor
- steuer queries the database to find the pick order associated with the container number
- out of the information about requested articles, steuer determines the source containers that could be used to fulfill the order and requests those containers from spider

Orders with orderstart

If WCS does not send us the target containers (e.g. they could be manually placed by the picker at the picking station), SRC is responsible for starting an order. In that case, there is a separate "***os***" ("*orderstart*") process which starts the pick orders.

Fetching and pre-fetching

steuer retrieves source containers that are required for fulfilling pick orders from the OSR. In order to determine the number of containers it requests from the OSR at the same time, there are two values in the configuration (set per picking station):

- "**fetch**": this value indicates the number of containers that can be ejected to the source loop at the same time. The value is constrained by the maximum number of containers that can fit between a diversion to the picking station and the picking station itself (including the source tray bays); when ejecting more containers, they would not fit and therefore travel additional rounds on the loop.
- "**prefetch**": this value represents the number of all the containers that have to be prepared to be sent to the picking stations. As it also includes the already fetched containers, the value is higher than the ***fetch*** value. Containers that are prepared, but not yet ejected are stored on the intermediate lift buffers.

Dynamic fetching

It is possible to select one of several different *fetch modes* – this can theoretically be done for each picking station individually (and even per order type on the picking station). That selection happens in the constructor function for the picking stations when instantiating the `OSR.SteuerCfg.FetchConfig` class.

While in older versions, the "*regular*" fetch was standard, nowadays the so-called "*dynamic fetch*" is used by default. This serves to optimize the container flow: with the regular fetch implementation, steuer will always retrieve the exact number of containers which is defined with the ***fetch*** value – no matter if the picking station is able to handle them or not.

The dynamic fetch, on the other hand, takes into account the currently handled orders at a picking station, and will only request as many containers as are required to fulfill those orders. So with dynamic fetching, the ***fetch*** and ***prefetch*** values are the maximum numbers, not strict quantities.

Target container handling

As with source containers, it is also possible to define how many target containers can fit on the conveyor belt between the diversion and the actual picking station, so as to not send too many at the same time to a specific picking station. This setting is called "***overload***", and is also defined per picking station.

Configuration

General

The steuer configuration file is structured as follows: first, all required libraries are imported. Secondly, several helper functions are defined which assist in creating the necessary configuration items listed below. Usually, these helper functions are not to be changed; rather you put all required parameters in the respective function calls. Then, the class `OSR.SteuerCfg.Config` is instantiated which creates the complete steuer config; this config is then written to the GCS. Finally, there is a call to a function which will historize the executed configuration file.

Please note that there have only recently been some efforts to standardize the steuer config file (starting with version 6.2); if you work with older configs, the files may be less consistent between projects. The configuration examples below are taken from a 6.3 version and show the most important parameters that need to be defined for the `OSR.SteuerCfg.Config` class instantiation.

Aisles and stations

towerMFSequence

This Python list defines all the aisles that are reachable by steuer. The abbreviation “MF” stands for “*multiple floors*” – in older versions (which only supported single floors), there was a “*towerSequence*” instead.

When defining the “towers”, you need to indicate the number for each aisle, as well as the specified “*fill grade*”, i.e. the desired percentage up to which all the storage locations in an aisle can be filled. Please note that especially in the case of multi-deep storage, we don’t set the fill grade to 100 %; this allows still for relocation operations to be carried through.

```
...
towerMFSequence = [
   getTower(1, 99.5),
   getTower(2, 99.5),
],
...
```

pickStationSequence

In this list, all existing picking stations have to be defined. In the function call to create the picking stations, the required parameters are the ID of the picking station as well as settings for the number of containers that each station can handle (i.e. "*overload*" for target containers and *fetch* and *prefetch* values for the source containers – for the latter values it's also possible to define different settings for the different modes of a picking station).

```
                                #psId, overload, fetch, prefetch, fetch_merge,
prefetch_merge
ps1 = createDefaultPickStationConfig(1101,  5,       6,     12,       10,
20)
ps2 = createDefaultPickStationConfig(1102,  5,       6,     12,       10,
20)
```

[…]

```
  ...
  pickStationSequence = [ps1, ps2],
  ...
```

loopSwitchSequence

All existing loop switches must be defined in this Python list. In addition to the PLC station ID, different parameters can be defined like which route (see below in the Floor layout section) a container shall take when it is unknown or if there is a reading error (here, the parameter is an index for the correct route), or whether or not the station has a height or weight check (these are Boolean values), and so on.

```
  ...
    loopSwitchSequence = [
                          #stationId, readingError, unknown, heightCheck,
weightCheck, roundRobin
        getLoopSwitchConfig(12400,     0,            1,       0,           0,
0),
        getLoopSwitchConfig(12500,     0,            0,       1,           0,
0),
        getLoopSwitchConfig(12501,     0,            0,       1,           0,
0),
    ],
  ...
```

exitStationSequence

This setting holds a list of all possible exits. There is a default exit which is called the ***SRC_EXIT*** and assigned a station number of ***91***. It is possible, though, to configure additional exits, e.g. for empty containers etc. In this case, you define additional exits (with consecutive numbers) in this list; yet you also have to define an assignment in the **TRANSPORTER_LOCATIONS** table in the database (relevant column name is ***TL_X***, and the transporter needs to be steuer)!

```
  exitStationSequence = [91, 92],
```

The output below shows a query on the **TRANSPORTER_LOCATIONS** joined with **LOCATIONS** and **TRANSPORTERS** for the exits from the example above:

```
      TL_X LOC_EXTERNAL_CODE                    TRANSPORTER_NAME
---------- ------------------------------------ --------------------------------
        91 SRC_EXIT                             STEUER
        91 SHUTTLE_EXIT                         STEUER
```

genericDiverterSequence

As there were different diverter types in use in older versions – to fulfill different functions – there are still several `xxxDiverterSequence` entries in the code, but we basically just use the `genericDiverterSequence` now to list all the PLC diverter stations on the target conveyor. Besides the station number, there are again parameters that can be specified like the route index for unknown containers or containers with reading errors.

```
  ...
  genericDiverterSequence = [
                        #(station, clamp, read, unk)
     createGenericDiverter(11700,   1,     0,    0),
     createGenericDiverter(11701,   0,     0,    0),
  ],
  ...
```

beltDiverterSequence

The station numbers of all of the created diverters (whether in the `genericDiverterSequence` or any other) need to be added to this list.

```
  ...
  beltDiverterSequence = [11700, 11701],
  ...
```

Layout and routing

layout

This list contains the full structure of the prezone system, i.e. if there is more than one floor, all floors are defined here with their respective names and IDs. Please note that these entries need to correspond with entries in the database (**LOCATIONS** table); in general, the pool config is done first in the WH model config (which is the basis for the data in the database), and then reflected in the steuer configuration.

If there are several floors, the database ID of the first one will be usually be "**103**". In case you have a small warehouse with only one floor, and you use the entry generated by default ("**POOL**" with ID 1 – which is not explicitly set in the WH model), you will use that combination in the code piece below. Anyways, the correct approach is as follows:

- configure the warehouse model
- create the OSR instance
- check in the OSR instance for the IDs of the created pools
- use the correct pool name and ID combination in the steuer config

Underneath the floor definition, the next hierarchical structure that needs to be defined is the segment; even if there is no segmentation in your project, there still needs to be one segment defined.

Within the segment definition then all accessible aisles/towers, picking stations, fill readers, loop switches and so on need to be listed as well as fetch and error stations.

```
...
  layout = [
      OSR.SteuerCfg.Floor(
          id = 1,
          name = "POOL",
          segments = [
              OSR.SteuerCfg.Segment(
                  id = 0,
                  zoneName = "",
                  towerRefSequence = [1, 2],
                  pickStationRefSequence = [1103, 1101, 1102],
                  fillReaderRefSequence = [12400],
                  loopSwitchRefSequence = [12500, 12501],
                  scaleStationRefSequence = [],
                  fetchStationSequence = [],
                  errorStationSequence = [
                      OSR.SteuerCfg.ErrorStationStruct(
                          id = 1101,
```

```
                            errorTrayTargetLocation = "",
                            errorTypes = [OSR.SteuerCfg.ALL]
                        )
                    ]
                )
            ],
            segmentsReachable = 1
        )
    ],
    ...
```

Below, 2 sequences from the above code are explained in detail:

fetchStationSequence

In this list we define all stations that are defined as "*fetch stations*", i.e. where containers from the OSR are considered to be fetched.

errorStationSequence

This list contains all (picking) stations that serve as error stations – i.e. where containers that trigger a specific error are sent to for handling. These errors can be a height error, a weight error, or unknown container error. It's possible to assign different stations for different types of errors, so to send overweight containers to one picking station, and unknown containers to a different picking station (or an exit). It's also possible – as in the example above – to use `OSR.SteuerCfg.ALL` as a wildcard for all errors.

loopLayout

A little bit misleadingly named, this list basically holds the routing configuration of steuer – it is an enumeration of all stations where diversions can take place, with definitions of all possible routes each (instances of `OSR.SteuerCfg.RouteEntry`).

For each and every station and their routes, you define:

- the direction (like "**0**", "**-1**") for a route
- a list of stations that can be reached when following that direction. Instead of listing all stations, it's also possible to use a shortcut entry like "**FP1:0**" which translates to the following:
  - "**F**" stands for "*all fill readers*", "**P**" means "*all picking stations*" (another shorthand notation would be "**T**" for "*all towers*")
  - "**1**" is the ID of the floor (so in our example above, it is the "***POOL***")
  - the "**0"** after the colon indicates the index of the segment
  - so in complete, this reads as: "*reach all fill readers and picking stations on the floor with ID 1 and segment with ID 0*"

- the next station where steuer shall send a pre-announcement of the container; “**0**” means that there is no pre-announcing

When a container is clamped at a station, steuer looks at the target destination and determines then based on the route entries which direction to choose.

Please note: If the designated target for a container is not listed in the routing table of a station, steuer will send the container to the default direction (entry “**0**”), so it’s actually not mandatory to list the targets of the default route; this is for better readability.

```
...
    loopLayout = [
        # =========== POOL ========================
        OSR.SteuerCfg.Routes( 12500, [
            OSR.SteuerCfg.RouteEntry(  0, ["FP1:0"], 0),
            OSR.SteuerCfg.RouteEntry( -1, ["1"], 0)
        ]),
...
        OSR.SteuerCfg.Routes( 12400,[
            OSR.SteuerCfg.RouteEntry(  0, ["TPF1:0"], 0),
            OSR.SteuerCfg.RouteEntry( -1, ["91"], 91)
        ]),
...
        # =========== TARGET CONVEYOR ========================

        OSR.SteuerCfg.Routes(90, [
            OSR.SteuerCfg.RouteEntry(  0, ["11702"], 0)
        ]),

        OSR.SteuerCfg.Routes(11702, [
            OSR.SteuerCfg.RouteEntry(  0, ["1101","1102"], 11700),
            OSR.SteuerCfg.RouteEntry( -1, ["1103"], 1103)
        ]),
...
        # =========== PICKSTATIONS ========================
        OSR.SteuerCfg.Routes(1103, [
            OSR.SteuerCfg.RouteEntry(  0, ["91"], 91)
        ]),
...
```

Process startup

As all the other processes of SRC, steuer is started through a wrapper `run` shell script which is located underneath steuer's subdirectory within the `_service` folder of the OSR instance.

As can be seen from its run file, steuer will wait for the spider process to be up for some time already before it assumes its activities.

Also, what is important to note is that steuer will not work correctly if it cannot reach the configured opa processes for the picking stations. The opa processes in turn will only start up if they can reach the physical stations once. Therefore, when preparing a project, and you want to start up the processes, remove any entries from the `pickStationSequence`.

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	Adaptations for SRC 6.3; minor changes
1.0	Initial version


KNAPP

steuer

what is it for

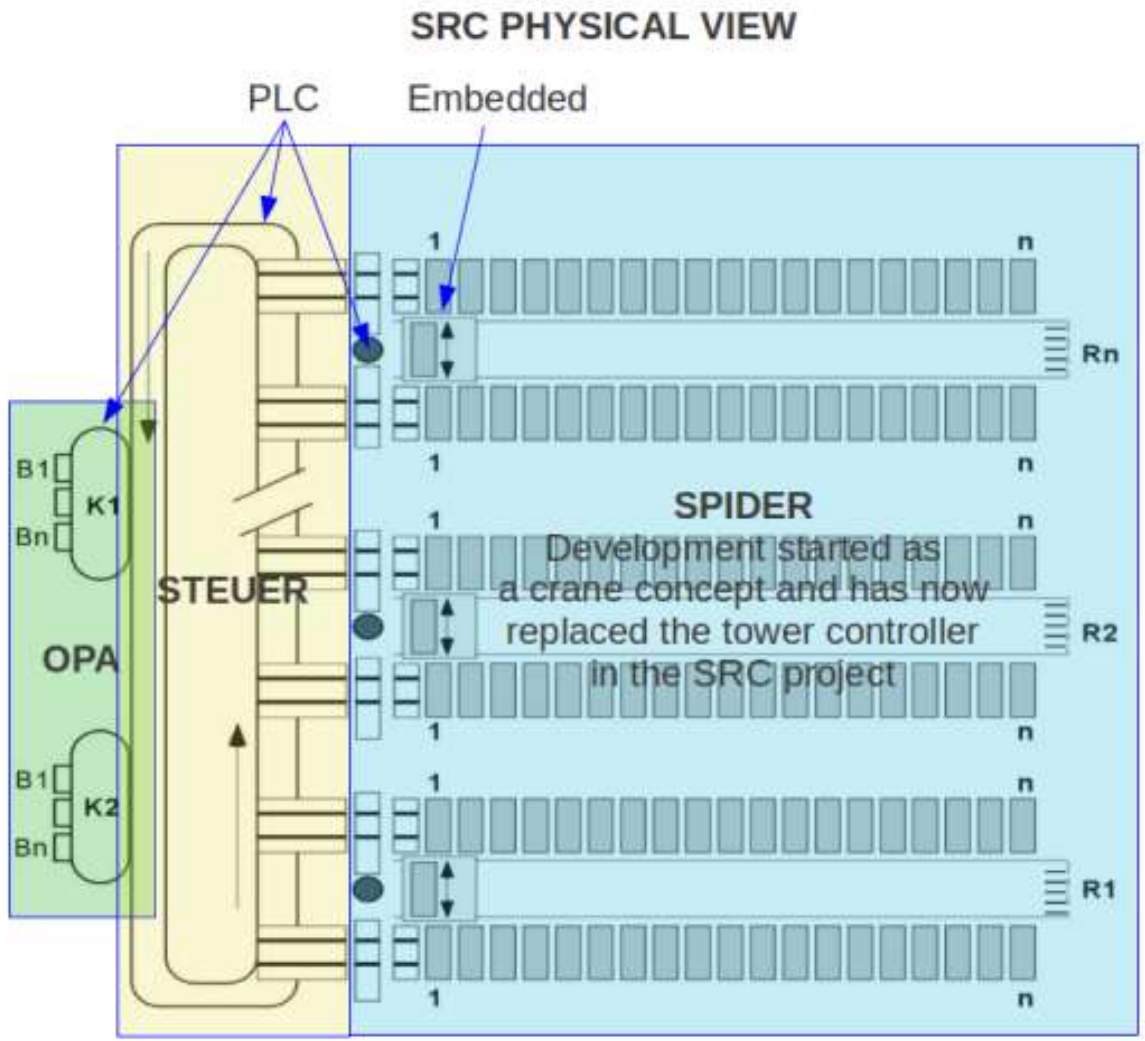
controls most parts of prezone

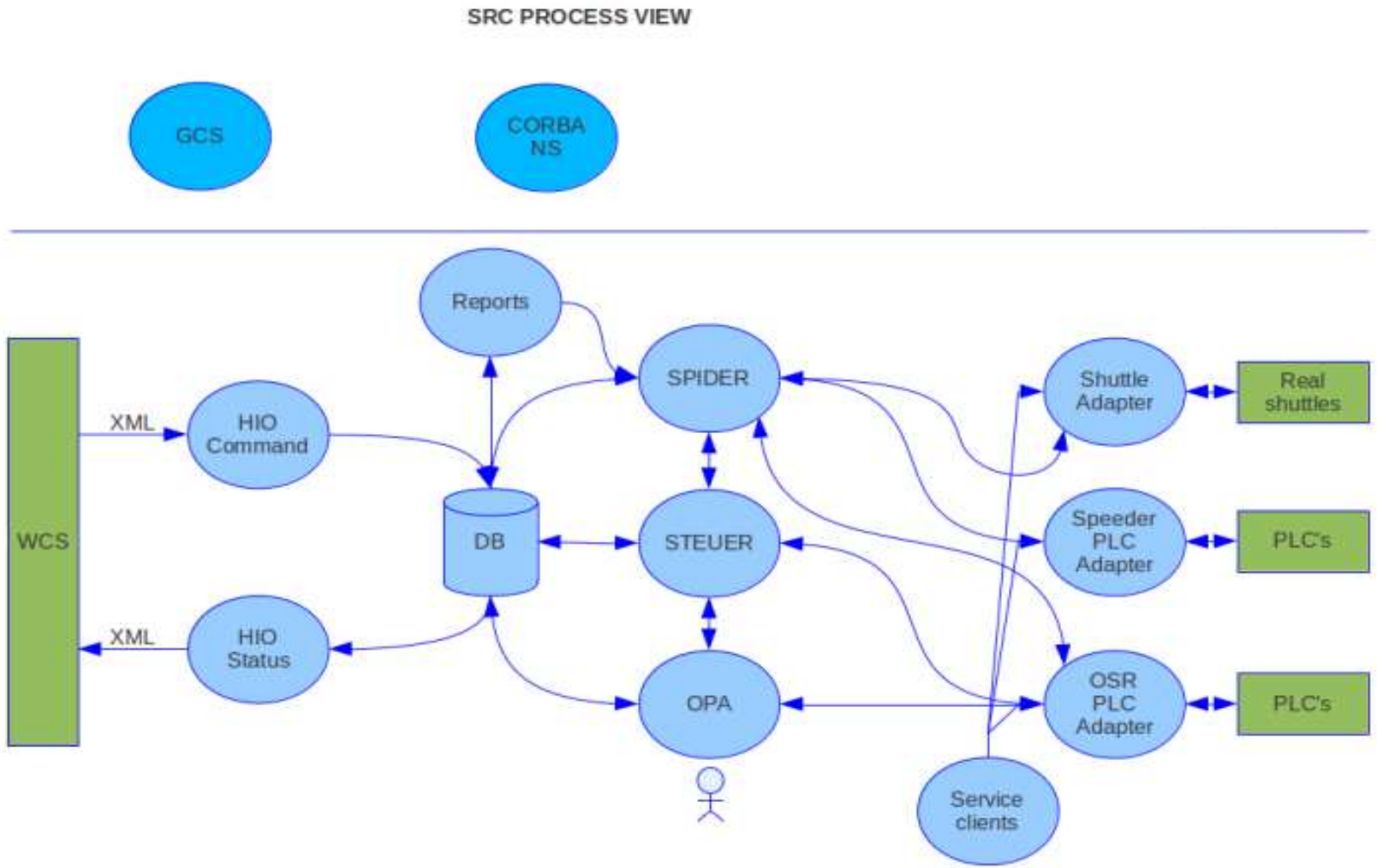
stores trays to OSR

handles pick orders

coordinates between stations

x) prezone stations
x) pick stations
x) OSR aisles


SRC PHYSICAL VIEW
PLC
Embedded
B1
K1
Bn
STEUER
OPA
B1
K2
Bn
1
n
Rn
1
n
SPIDER
Development started as
a crane concept and has now
replaced the tower controller
in the SRC project
1
n
R2
1
n
1
n
R1
1
n


SRC PROCESS VIEW
GCS
CORBA NS
Reports
SPIDER
Shuttle Adapter
Real shuttles
XML
HIO Command
WCS
DB
STEUER
Speeder PLC Adapter
PLC's
XML
HIO Status
OPA
OSR PLC Adapter
PLC's
Service clients

steuer and pick orders

x) receives and starts pick orders

x) sends target trays to pick station

x) requests source trays from OSR

x) sends source trays to pick stations

source trays

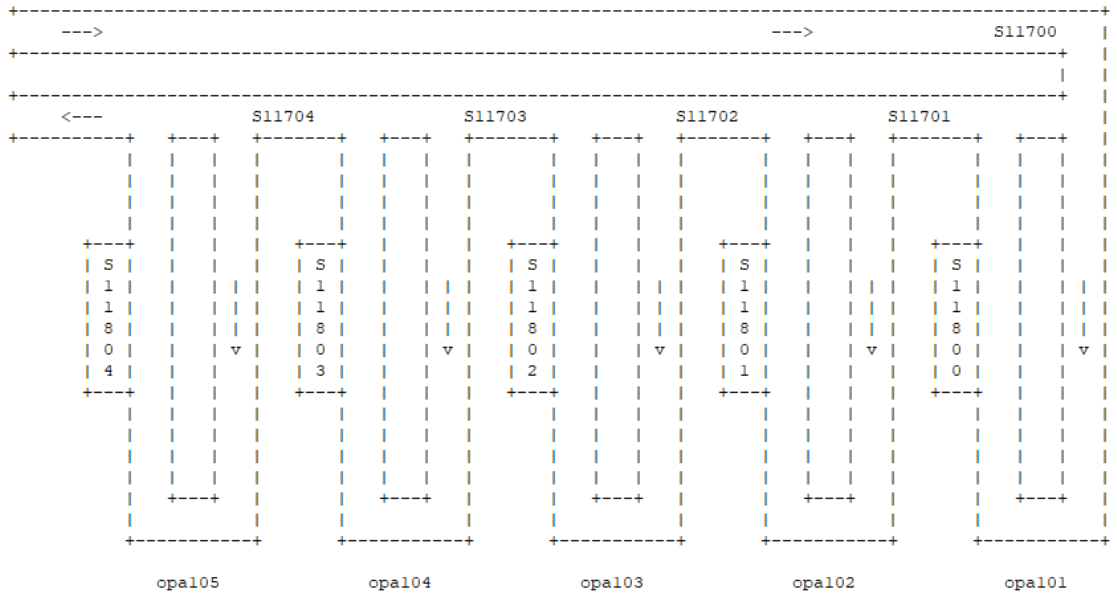
known to database (table containers)

controlled by LoopSwitches

can have tasks to targets

closed conveyor loops

target containers

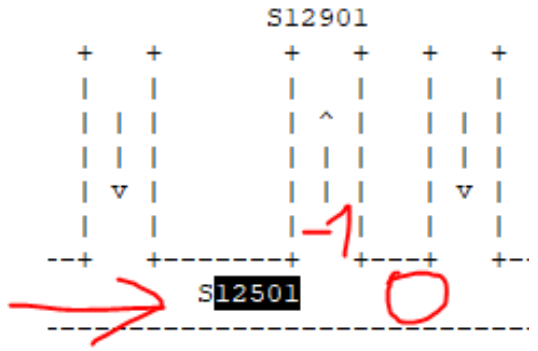


controlled by diverters

GenericDiverter

usually no possibility to loop back to first station

plc communication




```
2021-04-20 16:38:11.126 INFO  [S12501] <-- containerClamped(CID: 10001919, UNK: 1, RE: 0, HF: 0, SF: 0)
2021-04-20 16:38:11.127 INFO  [S12501] --> announceContainer(CID: 10001919, SRC: 0, DEST: -1, HEIGHT: 1)
2021-04-20 16:38:12.960 INFO  [S12501] <-- containerProcessed(CID: 10001919, POS: -1, UNK: 0, RE: 0, HF: 0, SF: 0,
OOW: 0, OH: 0, OC: 0, NB: 0, TU: 0)
2021-04-20 16:38:12.961 INFO  [S12501] --> deleteContainer(CID: 10001919)

-------------------------------------------------------------------------------------------------------

2021-04-20 16:38:11.126 INFO  [Switch-12501] Reimpl: containerClamped(10001919,U=true,R=false,H=false,F=false)
2021-04-20 16:38:11.126 INFO  [Switch-12501] Exec: containerClamped(10001919,U=true,R=false,H=false,F=false, sourceId=0)
…
2021-04-20 16:38:11.127 INFO  [Switch-12501] Pumping: announceContainer(10001919,-1,h=1,platformId=0)
…
…
2021-04-20 16:38:12.960 INFO  [Switch-12501] Reimpl: containerProcessed(10001919,U=false,R=false,H=false,F=false,W=false,P=-1)
2021-04-20 16:38:12.960 INFO  [Switch-12501] Exec: containerProcessed(10001919,U=false,R=false,H=false,F=false,W=false,P=-1)
….
2021-04-20 16:38:12.961 INFO  [Switch-12501] Pumping: deleteContainer(10001919)
```

enqueuing/dequeuing

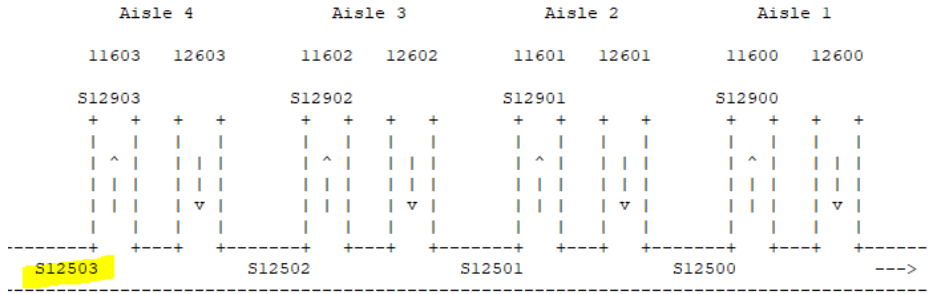
```
2021-04-19 15:00:00.330 INFO  [GenericDiverter-13703] Reimpl:
containerProcessed(200084583,U=false,R=false,H=false,F=false,r=false,W=false,P=0)
2021-04-19 15:00:00.332 INFO  [GenericDiverter-13704] Reimpl:
containerProcessed(200004273,U=false,R=false,H=false,F=false,r=false,W=false,P=0)
2021-04-19 15:00:00.335 INFO  [GenericDiverter-13703] Reimpl:
containerClamped(200054976,U=true,R=false,H=false,F=false)

2021-04-19 15:00:00.336 INFO  [Switch-12400] Reimpl:
containerProcessed(10008485,U=false,R=false,H=false,F=false,W=false,P=0)

2021-04-19 15:00:00.336 INFO  [Switch-12400] Exec:
containerProcessed(10008485,U=false,R=false,H=false,F=false,W=false,P=0)
```

steuer choosing direction

Tasks(Target(106)::PICK(4889, 4449, 101717,),)
x) contains target
x) contains type

```
2021-04-19 17:46:58.294 INFO  [Switch-12518] Reimpl: containerClamped(101717,U=true,R=false,H=false,F=false)

2021-04-19 17:46:58.294 INFO  [Switch-12518] Exec: containerClamped(101717,U=true,R=false,H=false,F=false,
sourceId=0)


2021-04-19 17:46:58.294 INFO  [Switch-12518] preannouncing ... 101717

2021-04-19 17:46:58.294 INFO  [Switch-12518] Tray(101717 FULL_TRAY ToProcess height:null, weight:null, homeZone:1,
sSeg:0, floor:1800, sTwr:18, dTwr:null, cTwr:18, lastSeen:12518) :Tasks( Target(106)

::PICK(4889, 4449, 101717, ),  )

2021-04-19 17:46:58.294 DEBUG [Switch-12518] temp target for home segment: 106

2021-04-19 17:46:58.294 INFO  [Switch-12518] Pumping: announceContainer(101717,0,h=1,platformId=0)
```

fill reader



chooses aisle and announces tray to spider

```
2021-04-19 16:24:27.316 INFO  [Switch-12503] Reimpl: containerClamped(10026830,U=true,R=false,H=false,F=false)
2021-04-19 16:24:27.316 INFO  [Switch-12503] Exec: containerClamped(10026830,U=true,R=false,H=false,F=false,
sourceId=0)
2021-04-19 16:24:27.316 DEBUG [Switch-12503] I'm a matching filler.
2021-04-19 16:24:27.316 DEBUG [Tray] reserveTower(z:1,s:0)
2021-04-19 16:24:27.317 DEBUG [StorageSystem] selectTowerOnFloorForTray(tray: 10026830, floor: 0, zone: 1, segIndex:
1, avoid: null)
2021-04-19 16:24:27.317 DEBUG [TowerSelector] getNextTarget( 1, 1, null ) called
2021-04-19 16:24:27.317 INFO  [TowerSelector] getNextTarget() returns: 3
2021-04-19 16:24:27.317 INFO  [TowerMF-3] Pumping: announceToZone(10026830, zone: PRODUCT_STORAGE_AISLE_3)
```

pick station mode changes

```
2021-04-21 07:46:54.566 INFO  [PickStation-103] Reimpl:
modeChanged(true,false,false,false,false,false,false,false,false,false,false)
2021-04-21 07:46:54.566 INFO  [PickStation-103] Exec:
modeChanged(true,false,false,false,false,false,false,false,false,false,false)
2021-04-21 07:46:54.566 INFO  [PICK] starting PICK mode.


2021-04-21 08:46:59.638 INFO  [PickStation-103] Reimpl:
modeChanged(false,false,false,false,false,false,false,false,false,false,false)
2021-04-21 08:46:59.638 INFO  [PickStation-103] Exec:
modeChanged(false,false,false,false,false,false,false,false,false,false,false)
2021-04-21 08:46:59.638 INFO  [PICK] ending PICK mode.
```

starting an order

```
2021-04-19 14:59:34.796 INFO  [Entrance-90] Reimpl: containerProcessed(2332263,U=true,R=false,H=false,F=false,r=false,W=false,P=0)
2021-04-19 14:59:34.796 INFO  [Entrance-90] Exec: containerProcessed(2332263,U=true,R=false,H=false,F=false,r=false,W=false,P=0)


2021-04-19 15:00:01.288 INFO  [GenericDiverter-11700] Exec: containerClamped(2332263,U=true,R=false,H=false,F=false)

2021-04-19 15:00:01.289 INFO  [TPC] startAndAnnounce( 2332263, PS:0, PoolLocId: null, (null), split=false)
2021-04-19 15:00:01.289 DEBUG [TPC] startNow( zone: 0 )
2021-04-19 15:00:01.289 INFO  [DBPickOrder] startNow( zone: 0 psId: 0 poolLocId:null ) called

2021-04-19 15:00:01.344 INFO  [DBPickOrder] PickOrder(411826).loadLineList()
2021-04-19 15:00:01.347 DEBUG [DBPickOrder] DBPickOrder( 411826, null, null,
  Line( DBPickOrderLine( id=1256182, porId=411826, state=0, trayId=10012973, container=null, targetSlot=1, processingSequence=null, serialPicked=null) )
  Line( DBPickOrderLine( id=1256185, porId=411826, state=0, trayId=10014016, container=null, targetSlot=1, processingSequence=null, serialPicked=null) )
  Line( DBPickOrderLine( id=1256183, porId=411826, state=0, trayId=10022256, container=null, targetSlot=1, processingSequence=null, serialPicked=null) )
  Line( DBPickOrderLine( id=1256184, porId=411826, state=0, trayId=10026104, container=null, targetSlot=1, processingSequence=null, serialPicked=null) )
 )

2021-04-19 15:00:01.347 DEBUG [TPC] TPC(2332263) TrayList.size() = 4
2021-04-19 15:00:01.347 INFO  [SaturationSortPickStationSelectionStrategy] Do: SaturationSortPickStationSelectionStrategy
...
2021-04-19 15:00:01.347 INFO  [GenericDiverter-11700] Pumping: announceContainer(2332263,-1)
2021-04-19 15:00:01.347 INFO  [GenericDiverter-11700] put TPC(2332263) in announcement of all stations till target PS!
2021-04-19 15:00:01.347 DEBUG [GenericDiverter-11700] preAnnounceForPS( 101, 2332263,pos=-1 )  in clamp:

2021-04-19 15:03:04.132 INFO  [PickStation-101] Pumping: announcePickOrder( 2332263, 1, [ PICK(411826, 1256182, 10012973, 2332263), PICK(411826, 1256185,
10014016, 2332263), PICK(411826, 1256183, 100
```

trayList

information about trays which are assigned to pick stations

x) order information

x) sequence information

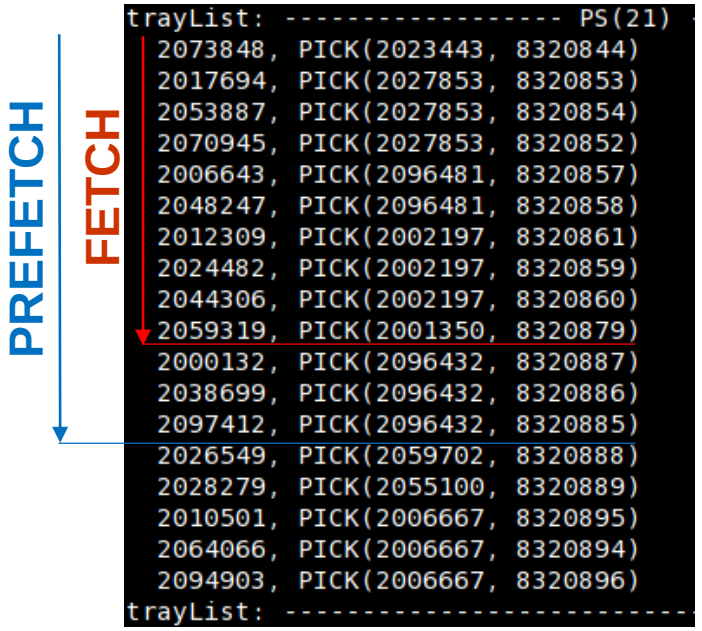
order id, line id, tray id, target id

```
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ------------ PS(103) --------->>>
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10018087, PICK(412255, 1257670, 10018087, 2332596) }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10001892, PICK(412270, 1257723, 10001892, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10027101, PICK(412270, 1257717, 10027101, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ------------------------------<<< size: 3
```

fetching trays

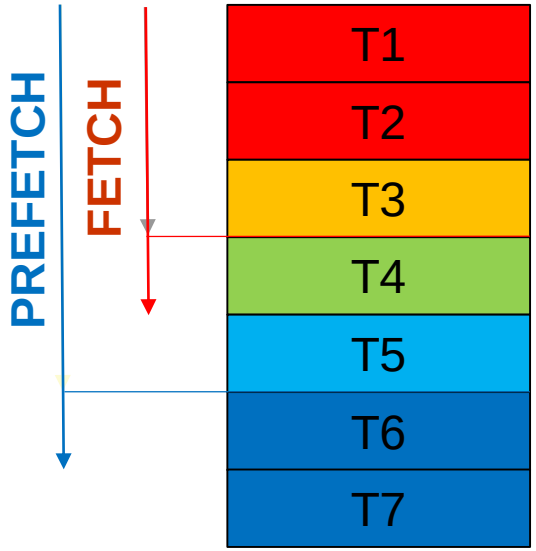
steuer requests tray from OSR

in this example:
fetch: 10
prefetch: 13




```
2021-04-19 16:00:04.621 INFO
[TowerMF-4] Pumping: prefetchToZone(10025213, zone: POOL, psId: 103, orderId: 412171, fetchSeq: 39)

2021-04-19 16:01:57.379 INFO
[TowerMF-4] Pumping: ejectToZone(10025213, zone: POOL, psId: 103, orderId: 412171, fetchSeq: 25)

2021-04-19 16:02:04.091 INFO
[TowerMF-4] Exec: containerEjectedToZone(10025213,nB=false,U=false,FLOOR=POOL,fromOffline=false)
```

dynamic fetch



dynamically increasing or decreasing
fetch values based on order size and sequence

PowerPoint presentation

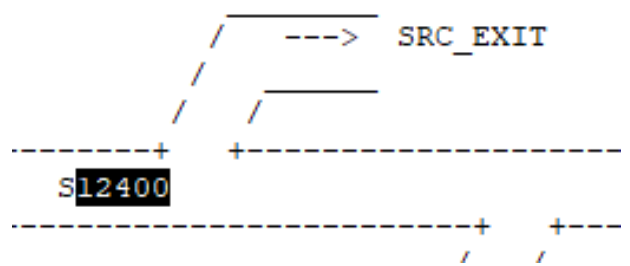
inventory orders

```
2021-04-12 17:24:43.045 INFO  [PickStation-11:1851] Exec:
modeChanged(false,false,false,true,false,false,false,false,false,false,false)
2021-04-12 17:24:43.046 INFO  [Inventory-PS11:57] starting Inventory-PS11 mode.

2021-04-12 17:25:37.331 INFO  [Inventory-PS11:89] getNext( PS: 11, PoolLocId: 204 )
2021-04-12 17:25:37.331 DEBUG [DBInventory:57] at.syslog.osr.db.Inventory.getNextTray()
2021-04-12 17:25:37.366 DEBUG [Tray:473] Tray(2084523287).addTypeUniqueTask(11, INVENTORY(0, 0, 2084523287, ))

2021-04-12 17:25:37.381 INFO  [TowerMF-1:676] Pumping: ejectToZone(2084523287, zone: POOL_1800, psId: 11, orderId: 0)

2021-04-12 17:25:45.048 INFO  [PickStation-11:2063] Pumping: announceInventoryTray(2084523287)
```

merge orders

```
2021-04-19 11:08:53.024 INFO  [PickStation-104] Exec:
modeChanged(false,false,false,false,false,true,false,false,false,false,false)
2021-04-19 11:08:53.024 INFO  [Merge] starting Merge mode.

2021-04-19 11:08:59.128 INFO  [Merge] get next MergeOrder
2021-04-19 11:08:59.128 DEBUG [MergeOrder] getNext( ps:104, TrayType.FULL_TRAY )

2021-04-19 11:08:59.540 INFO  [Merge] Found: MergeOrder( id:13041, trayList:  10006719 10019384  )
2021-04-19 11:08:59.540 INFO  [PickStation-104] Pumping: announceMergeOrder( 13041)
2021-04-19 11:08:59.540 DEBUG [Tray] Tray(10006719).addUniqueTask(104, MERGE(13041, 13099, 10006719, ))
2021-04-19 11:08:59.543 INFO  [TowerMF-2] Pumping: ejectToZone(10006719, zone: POOL, psId: 104, orderId: 13041,
fetchSeq: 1)
…
2021-04-19 11:08:59.543 DEBUG [Tray] Tray(10019384).addUniqueTask(104, MERGE(13041, 13099, 10019384, ))
2021-04-19 11:08:59.544 INFO  [TowerMF-4] Pumping: ejectToZone(10019384, zone: POOL, psId: 104, orderId: 13041,
fetchSeq: 2)
```

transport tasks



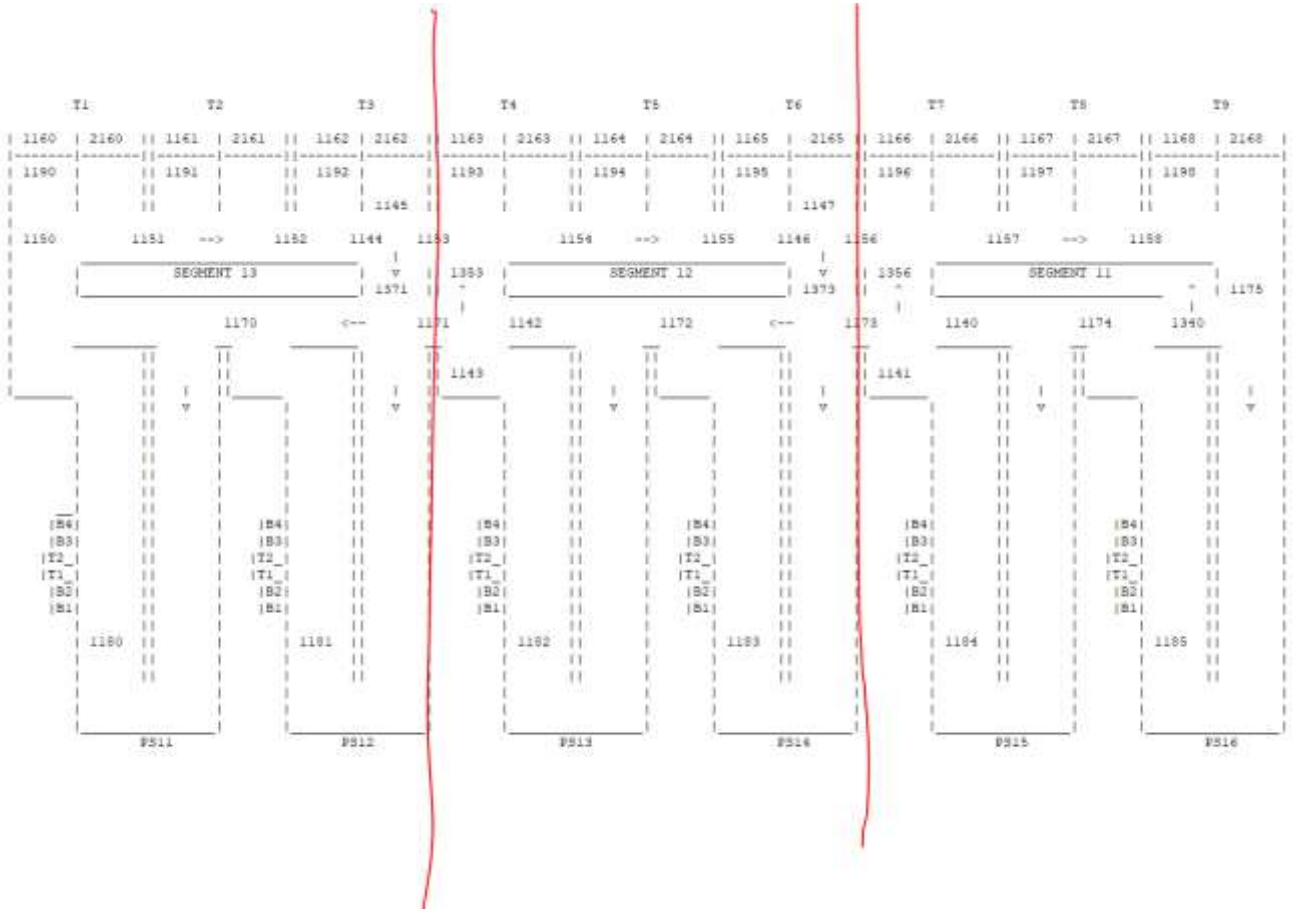

```
2021-04-19 11:08:40.637 INFO  [TransportTaskExecutor] Exec:
addTask( cid=10012615, target=91, orderId=2040772, lineId=2040773, system=SPIDER )


2021-04-19 11:09:28.874 INFO  [Switch-12400] Exec:
containerProcessed(10012615,U=false,R=false,H=false,F=false,W=false,P=-1)

2021-04-19 11:09:28.874 INFO  [Switch-12400] Tray(10012615 FULL_TRAY ToProcess height:null, weight:null, homeZone:1,
sSeg:0, floor:0, sTwr:3, dTwr:null, cTwr:3, lastSeen:12400) :Tasks( TransportTask( sys=SPIDER, target=91,
orderId=2040772, lineId=2040773 ),  )
2021-04-19 11:09:28.874 INFO  [Tray] Going to finish TransportTask( sys=SPIDER, target=91, orderId=2040772,
lineId=2040773 )
2021-04-19 11:09:28.874 DEBUG [TransportTask] finish TransportTask for 10012615
```

segmented systems

source trays stay
in segments

shorter distance

mirrored stock



steuer in orderstart systems

Work Package handling

```
2021-04-20 18:47:27.246 INFO  [OrderStarter:90] Received OrderStart WorkPackage Message.
2021-04-20 18:47:27.246 INFO  [PickStation-23:3192] Reimpl: workPackageReceieved( wpId: 43712 )
2021-04-20 18:47:27.246 INFO  [PickStation-23:3203] Exec: workPackageReceieved( wpId: 43712 )
2021-04-20 18:47:27.246 DEBUG [PickStation-23:3204] WorkPackage(
wpId=43712,orderId=1413628,type=MANUAL_PICK,target=23,orderContainerId=null,reannounce=false )
    Line( id=3691094,cid=11019512,source=5,type=HALF_TRAY,homeZone=null,seq=null )
2021-04-20 18:47:27.246 INFO  [PickStation-23:3228] Pumping: announceOrder( 1413628 )
```

Configuration

```
core/osr-6-2/java/at/syslog/osr/steuer/steuer_cfg.py
```

steuer_cfg.idl

x) shows structure of config file

x) useful comments

`/kisoft/user/osr/source/src/procs/osr/idl/steuer_cfg.idl`

`(PL 13: /kisoft/build/src/procs/osr/idl/steuer_cfg.idl)`

Towers

```
def getTower(id, fillLevel):
    return OSR.SteuerCfg.TowerMF(
        id = id,
        towerStorageZones = [
            OSR.SteuerCfg.TowerStorageZone("SBLOCK_%d_STORAGE" % id, "", fillLevel)
        ],
        towerFillLevel = fillLevel
    )
```

```
# towerMFSequence of double/twin lift towers
towerMFSequence = [
    getTower(1, 99.5),
    getTower(2, 99.5),
    getTower(3, 99.5),
    getTower(4, 99.5)
],
```

pick stations

```
# ========== PickStation definitions =========
                                    #psId, overload, fetch, prefetch, fetch_merge, prefetch_merge
ps1 = createDefaultPickStationConfig(101,  15,       18,    32,       10,          20)
ps2 = createDefaultPickStationConfig(102,  15,       18,    32,       10,          20)
ps3 = createDefaultPickStationConfig(103,  15,       18,    32,       10,          20)
ps4 = createDefaultPickStationConfig(104,  15,       18,    32,       10,          20)
ps5 = createDefaultPickStationConfig(105,  15,       18,    32,       10,          20)
```

```
    # this sequence MUST start with the first PickStation at the
    # conveyour, and end with the last.
    pickStationSequence = [ps1, ps2, ps3, ps4, ps5],
```

Loop switches

```
def getLoopSwitchConfig(stationId, readingError, unknown, heightCheck, weightCheck, roundRobin):
    return OSR.SteuerCfg.LoopSwitch(
        id = stationId,
        readingErrorRouteIndex = readingError,
        unknownContainerRouteIndex = unknown,
        removeTrayOnDiversionToExit = 0,
        cancelTransportTaskOnDiversionToWrongPosition = 0,
        hasHeightCheck = heightCheck,
        checkTrayWeight = weightCheck,
        uncheckedTrayWeightRouteIndex = 0,
        containerMovementConfigs = [],
        restrictedTowerSelectorRange = 0,
        roundRobinPossibleRoutes = roundRobin
    )
```

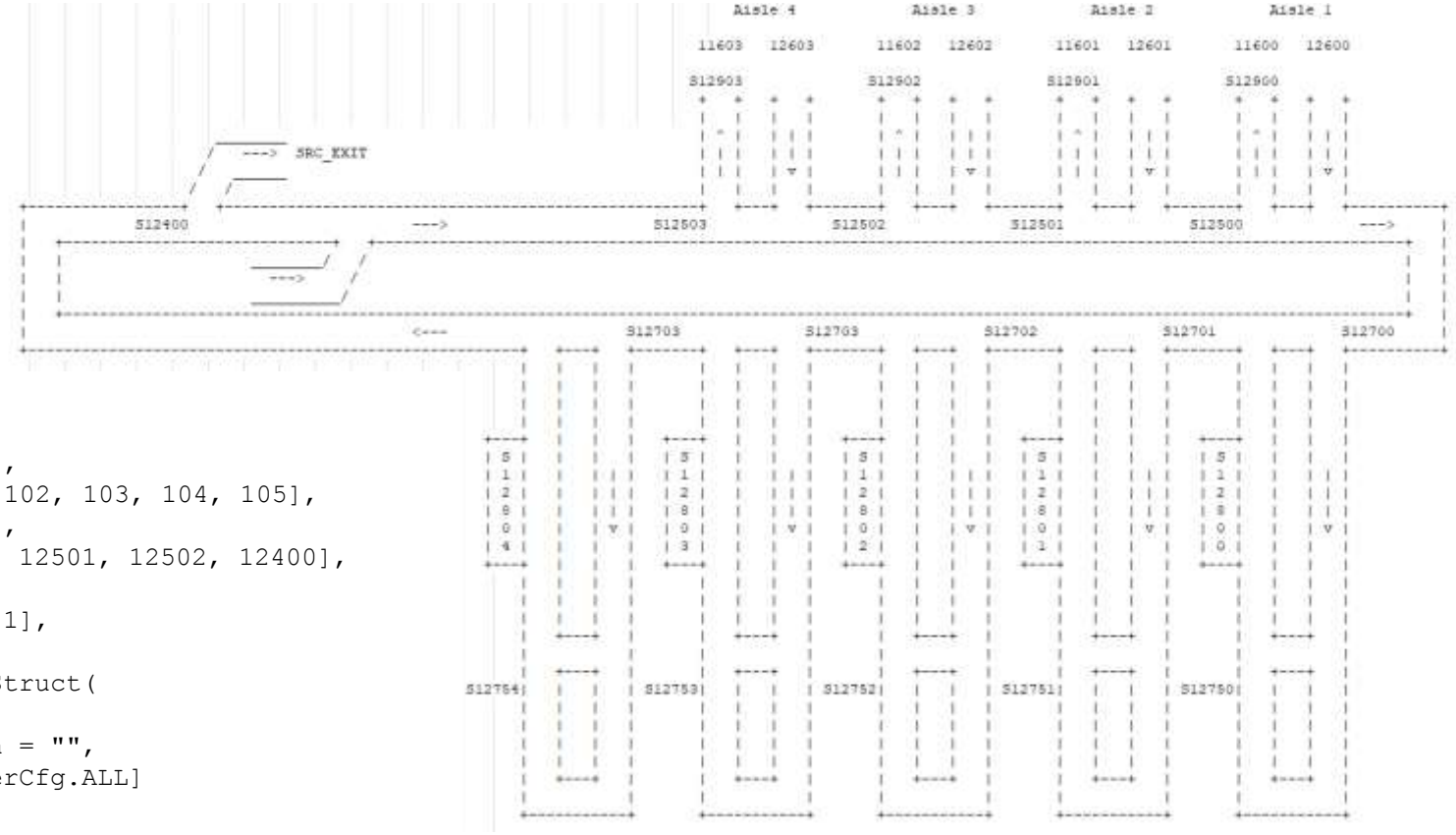
```
# the switches
loopSwitchSequence = [
                        #stationId, readingError, unknown, heightCheck, weightCheck, roundRobin
    getLoopSwitchConfig(12500,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(12501,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(12502,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(12503,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(12400,      1,            1,       0,           0,           0),

    getLoopSwitchConfig(13500,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(13501,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(13502,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(13503,      0,            0,       1,           0,           0),
    getLoopSwitchConfig(13400,      1,            1,       0,           0,           0)
],
```

Diverters

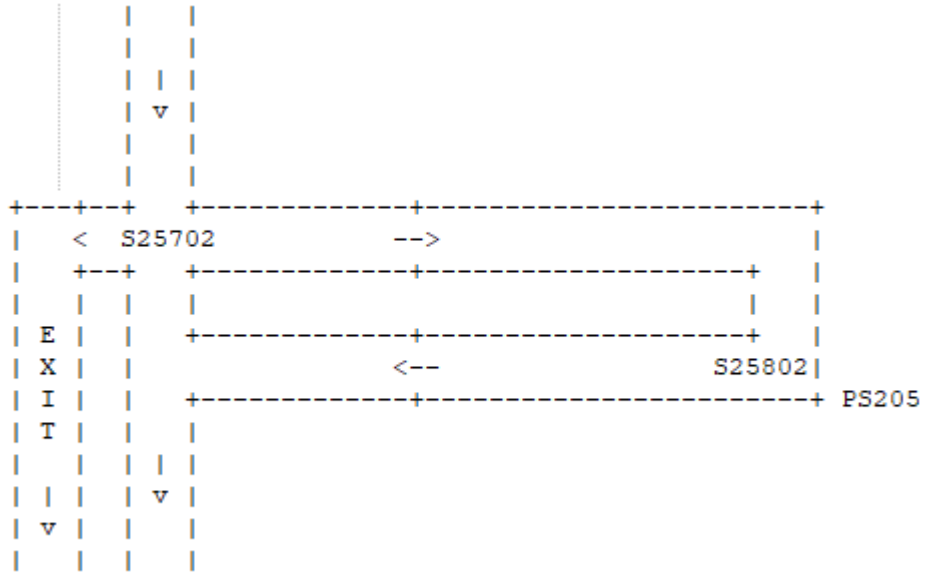
```
genericDiverterSequence = [
                          #(station, clamp, read, unk)
        createGenericDiverter(11700,   1,     0,    0),
        createGenericDiverter(11701,   0,     0,    0),
        createGenericDiverter(11702,   0,     0,    0),
        createGenericDiverter(11703,   0,     0,    0),
        createGenericDiverter(11704,   0,     0,    0)
    ],
```

Layout

```
layout = [
        OSR.SteuerCfg.Floor(
            id = 103,
            name = "POOL_1000",
            segments = [
                OSR.SteuerCfg.Segment(
                    id = 0,
                    zoneName = "",
                    towerRefSequence = [1, 2, 3, 4],
                    pickStationRefSequence = [101, 102, 103, 104, 105],
                    fillReaderRefSequence = [12503],
                    loopSwitchRefSequence = [12500, 12501, 12502, 12400],
                    scaleStationRefSequence = [],
                    fetchStationSequence = [12500, 1],
                    errorStationSequence = [
                        OSR.SteuerCfg.ErrorStationStruct(
                            id = 101,
                            errorTrayTargetLocation = "",
                            errorTypes = [OSR.SteuerCfg.ALL]
                        )
                    ]
                )
            ],
            segmentsReachable = 1
        ),
        OSR.SteuerCfg.Floor(
            id = 104,
            name = "POOL_2000",
            ...
```



virtual entrances

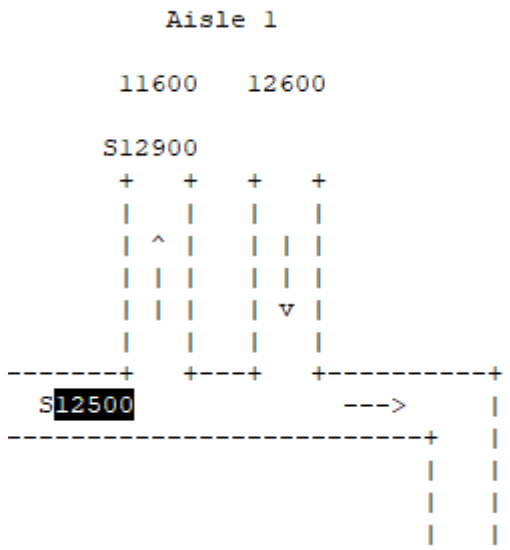



```
# the virtual Stations
virtualEntranceSequence = [
    getVirtualEntrance(925701, 205, [25702])
],
```

Routes

x) direction
x) possible targets
x) next2announce

F: fill reader, P: pick station, T: tower




```
OSR.SteuerCfg.Routes( 12500, [
    OSR.SteuerCfg.RouteEntry(  0, [„FP103:0"], 0),
    OSR.SteuerCfg.RouteEntry( -1, ["1", "93"], 0)
]),
```

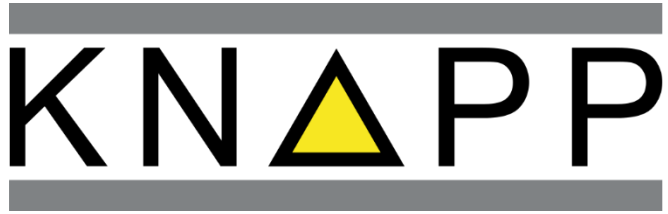
Thank you.

Vielen Dank.

Merci beaucoup.

Muchas gracias.

Mille grazie.



What is the steuer?

steuer in a nutshell

PURPOSE

- steuer – as one of the 3 main processes – is responsible for controlling the prezone
- It is responsible for the following functions:
  - Controlling the flow of containers in the prezone
  - Storing trays into the OSR
  - Handling pick orders
- It interfaces with and coordinates between:
  - PLC stations on the prezone
  - Picking stations
  - OSR aisles

Important terms

Terms

- **Sources / trays**
  - Are totes that are used for storage in the OSR
  - At the picking station, the totes goods are picked from
  - They are saved in the database (`CONTAINERS` table)
- **Targets / containers**
  - Totes that are used to pick goods into
- **Loop switches**
  - PLC stations on the source conveyors
- **Diverters**
  - PLC stations on the target conveyor system

Terms

- **Fill readers**
  - Usually last PLC station(s) on loop before aisle entry
- **Floors / segments**
  - Units for conveyor systems and designated areas within
- **Fetch / prefetch**
  - Definition of number of containers to be retrieved from OSR
- **Marriage**
  - Assignment of a container to an order in the system

steuer concepts

steuer and pick orders

DEFINITION

- steuer receives and starts pick orders
  - Trigger is arrival of target container
- It sends target containers to the picking stations
- It requests corresponding source trays from the OSR
- It sends the source trays to the picking stations

Sources and targets

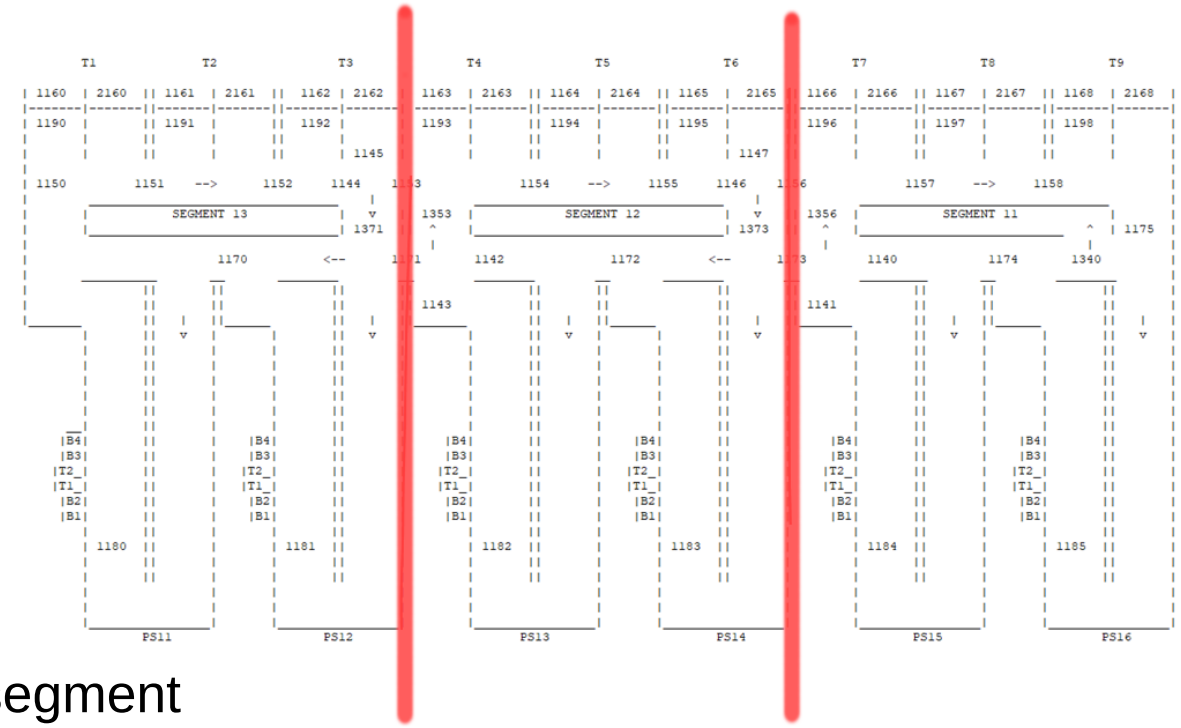
DEFINITION

- The prezone conveyor system consists of source and target conveyors
- Source:
  - The source line is usually implemented as a closed loop
  - It is connected to the OSR aisles
  - Totes on the source loop are the „trays“ – they are saved in the database (`CONTAINERS` table)
  - Source trays have tasks to targets
  - PLC stations on the source line are called loop switches
- Target:
  - The target conveyor is usually not implemented as a loop; when a container passes by the last picking station, there is no way to loop back
  - PLC stations on the target line are called diverters

Pool segmentation

DEFINITION

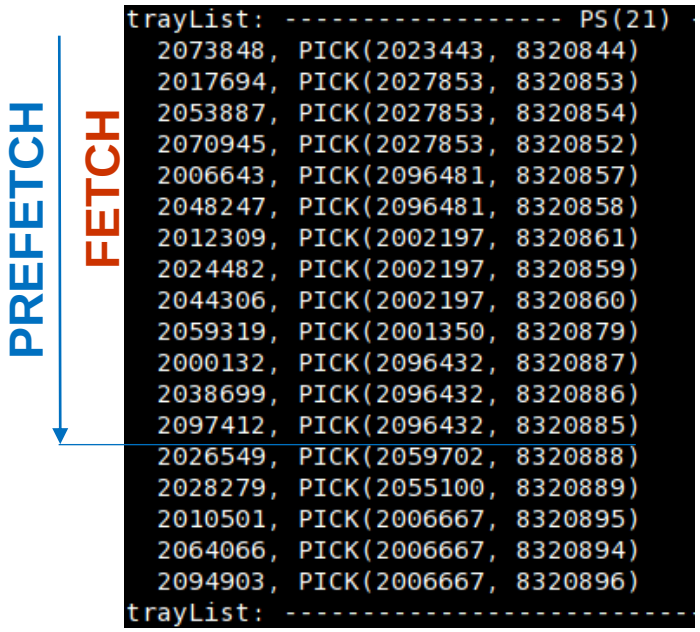
- In a non-segmented system, all OSR aisles connect to all picking stations
- Depending on the size of the conveyor system, this may result in long distances
- To shorten routes, it is possible to introduce segmentation
  - Additional diverters allow for taking shortcuts
- In such a scenario, trays will stay in their segment
- This requires mirrored stock in the OSR



Fetching and prefetching

DEFINITION

- In order to fulfil (pick) orders, steuer has a list of all source trays that are required
- steuer requests those trays from the OSR
- As the source conveyor system shall not get overloaded, not all of the trays in the list are requested at once
- Rather, we configure (per picking station) how many trays shall be
  - ejected to the loop (“fetch” parameter)
  - retrieved from the storage location and be kept on the lift buffers
- Lately, the so-called “dynamic fetch” has been implemented which replaces the fixed fetch and prefetch values with dynamically changing values (see specific presentation)



Tray list

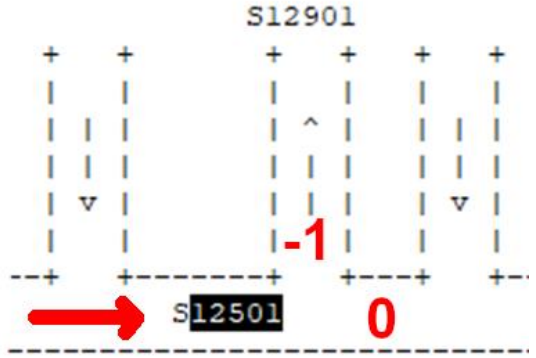
DEFINITION

- steuer maintains an internal list on which trays are assigned to picking stations
- This tray list may be displayed within the steuer logs, but it can also be shown by using the `steuer_status` command
- Displayed information (per picking station):
  - Tray number
  - Order information (order number, order line number, source and target container ID)
  - Status

```
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ------------ PS(103) --------->>>
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10018087, PICK(412255, 1257670, 10018087, 2332596) }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10001892, PICK(412270, 1257723, 10001892, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10027101, PICK(412270, 1257717, 10027101, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ----------------------------<<< size: 3
```

steuer communication & logs

steuer communication with plc



EXAMPLE

plc

```
2021-04-20 16:38:11.126 INFO  [S12501] <-- containerClamped(CID: 10001919, UNK: 1, RE: 0, HF: 0, SF: 0)
2021-04-20 16:38:11.127 INFO  [S12501] --> announceContainer(CID: 10001919, SRC: 0, DEST: -1, HEIGHT: 1)
2021-04-20 16:38:12.960 INFO  [S12501] <-- containerProcessed(CID: 10001919, POS: -1, UNK: 0, RE: 0, HF: 0, SF:
2021-04-20 16:38:12.961 INFO  [S12501] --> deleteContainer(CID: 10001919)
```

steuer

```
2021-04-20 16:38:11.126 INFO  [Switch-12501] Reimpl: containerClamped(10001919,U=true,R=false,H=false,F=false)
2021-04-20 16:38:11.126 INFO  [Switch-12501] Exec: containerClamped(10001919,U=true,R=false,H=false,F=false, sou
…
2021-04-20 16:38:11.127 INFO  [Switch-12501] Pumping: announceContainer(10001919,-1,h=1,platformId=0)
…
…
2021-04-20 16:38:12.960 INFO  [Switch-12501] Reimpl: containerProcessed(10001919,U=false,R=false,H=false,F=false
2021-04-20 16:38:12.960 INFO  [Switch-12501] Exec: containerProcessed(10001919,U=false,R=false,H=false,F=false,V
…
2021-04-20 16:38:12.961 INFO  [Switch-12501] Pumping: deleteContainer(10001919)
```

steuer choosing directions

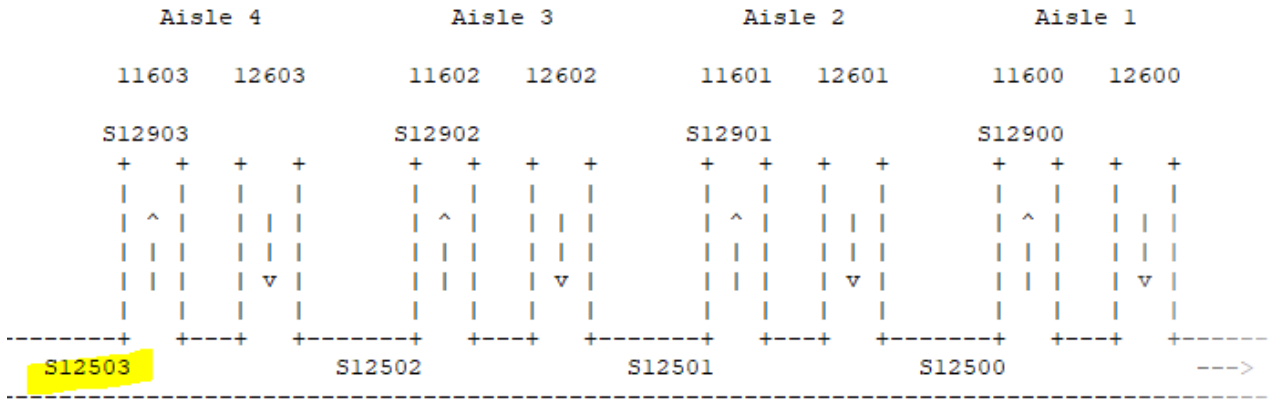
EXAMPLE

Tasks(Target(106)::PICK(4889, 4449, 101717,),)
x) contains target
x) contains type

```
2021-04-19 17:46:58.294 INFO  [Switch-12518] Reimpl:containerClamped(101717,U=true,R=false,H=false,F=false)
2021-04-19 17:46:58.294 INFO  [Switch-12518] Exec: containerClamped(101717,U=true,R=false,H=false,F=false,source
…
2021-04-19 17:46:58.294 INFO  [Switch-12518] preannouncing ... 101717
2021-04-19 17:46:58.294 INFO  [Switch-12518] Tray(101717 FULL_TRAY ToProcess height:null, weight:null,
 homeZone:1, sSeg:0, floor:1800, sTwr:18, dTwr:null, cTwr:18, lastSeen:12518) :Tasks( Target(106)
 ::PICK(4889, 4449, 101717, ),  )
2021-04-19 17:46:58.294 DEBUG [Switch-12518] temp target for home segment: 106
2021-04-19 17:46:58.294 INFO  [Switch-12518] Pumping: announceContainer(101717,0,h=1,platformId=0)
```

steuer choosing aisle

EXAMPLE



At the fill reader, steuer chooses an aisle and announces tray to spider

```
2021-04-19 16:24:27.316 INFO  [Switch-12503] Reimpl: containerClamped(10026830,U=true,R=false,H=false,F=false)
2021-04-19 16:24:27.316 INFO  [Switch-12503] Exec: containerClamped(10026830,U=true,R=false,H=false,F=false, sou
2021-04-19 16:24:27.316 DEBUG [Switch-12503] I'm a matching filler.
2021-04-19 16:24:27.316 DEBUG [Tray] reserveTower(z:1,s:0)
2021-04-19 16:24:27.317 DEBUG [StorageSystem] selectTowerOnFloorForTray(tray: 10026830, floor: 0, zone: 1, segIn
2021-04-19 16:24:27.317 DEBUG [TowerSelector] getNextTarget( 1, 1, null ) called
2021-04-19 16:24:27.317 INFO  [TowerSelector] getNextTarget() returns: 3
2021-04-19 16:24:27.317 INFO  [TowerMF-3] Pumping: announceToZone(10026830, zone: PRODUCT_STORAGE_AISLE_3)
```

Picking station mode changed

EXAMPLE

```
2021-04-21 07:46:54.566 INFO  [PickStation-103] Reimpl: modeChanged(true,false,false,false,false,false,false,fal
2021-04-21 07:46:54.566 INFO  [PickStation-103] Exec: modeChanged(true,false,false,false,false,false,false,false
2021-04-21 07:46:54.566 INFO  [PICK] starting PICK mode.

2021-04-21 08:46:59.638 INFO  [PickStation-103] Reimpl: modeChanged(false,false,false,false,false,false,false,fa
2021-04-21 08:46:59.638 INFO  [PickStation-103] Exec: modeChanged(false,false,false,false,false,false,false,fals
2021-04-21 08:46:59.638 INFO  [PICK] ending PICK mode.
```

Pick orders

EXAMPLE

```
2021-04-19 14:59:34.796 INFO  [Entrance-90] Exec: containerProcessed(2332263,U=true,R=false,H=false,F=false,r=fa
2021-04-19 15:00:01.288 INFO  [GenericDiverter-11700] Exec: containerClamped(2332263,U=true,R=false,H=false,F=fa
2021-04-19 15:00:01.289 INFO  [TPC] startAndAnnounce( 2332263, PS:0, PoolLocId: null, (null), split=false)
2021-04-19 15:00:01.289 INFO  [DBPickOrder] startNow( zone: 0 psId: 0 poolLocId:null ) called
2021-04-19 15:00:01.344 INFO  [DBPickOrder] PickOrder(411826).loadLineList()
2021-04-19 15:00:01.347 DEBUG [DBPickOrder] DBPickOrder( 411826, null, null,
  Line( DBPickOrderLine( id=1256182, porId=411826, state=0, trayId=10012973, container=null, targetSlot=1, proce
  Line( DBPickOrderLine( id=1256185, porId=411826, state=0, trayId=10014016, container=null, targetSlot=1, proce
  Line( DBPickOrderLine( id=1256183, porId=411826, state=0, trayId=10022256, container=null, targetSlot=1, proce
  Line( DBPickOrderLine( id=1256184, porId=411826, state=0, trayId=10026104, container=null, targetSlot=1, proce
 )
2021-04-19 15:00:01.347 DEBUG [TPC] TPC(2332263) TrayList.size() = 4
2021-04-19 15:00:01.347 INFO  [SaturationSortPickStationSelectionStrategy] Do: SaturationSortPickStationSelectio
2021-04-19 15:00:01.347 INFO  [GenericDiverter-11700] Pumping: announceContainer(2332263,-1)
2021-04-19 15:00:01.347 INFO  [GenericDiverter-11700] put TPC(2332263) in announcement of all stations till targ
2021-04-19 15:00:01.347 DEBUG [GenericDiverter-11700] preAnnounceForPS( 101, 2332263,pos=-1 )  in clamp:
2021-04-19 15:03:04.132 INFO  [PickStation-101] Pumping: announcePickOrder( 2332263, 1, [ PICK(411826, 1256182,
```

steuer in orderstart system

EXAMPLE

Work package handling

```
2021-04-20 18:47:27.246 INFO  [OrderStarter:90] Received OrderStart WorkPackage Message.
2021-04-20 18:47:27.246 INFO  [PickStation-23:3192] Reimpl: workPackageReceieved( wpId: 43712 )
2021-04-20 18:47:27.246 INFO  [PickStation-23:3203] Exec: workPackageReceieved( wpId: 43712 )
2021-04-20 18:47:27.246 DEBUG [PickStation-23:3204] WorkPackage(
wpId=43712,orderId=1413628,type=MANUAL_PICK,target=23,orderContainerId=null,reannounce=false )
    Line( id=3691094,cid=11019512,source=5,type=HALF_TRAY,homeZone=null,seq=null )
2021-04-20 18:47:27.246 INFO  [PickStation-23:3228] Pumping: announceOrder( 1413628 )
```

steuer fetching and prefetching trays

EXAMPLE

steuer requesting tray from OSR

in this example:
- fetch: 10
- prefetch: 13

```
2021-04-19 16:00:04.621 INFO [TowerMF-4] Pumping: prefetchToZone(10025213, zone: POOL, psId: 103, orderId:
412171, fetchSeq: 39)
…
2021-04-19 16:01:57.379 INFO [TowerMF-4] Pumping: ejectToZone(10025213, zone: POOL, psId: 103, orderId:
412171, fetchSeq: 25)
…
2021-04-19 16:02:04.091 INFO [TowerMF-4] Exec:
containerEjectedToZone(10025213,nB=false,U=false,FLOOR=POOL,fromOffline=false)
```

Inventory order

EXAMPLE

```
2021-04-12 17:24:43.045 INFO  [PickStation-11:1851] Exec: modeChanged(false,false,false,true,false,false,false,f
2021-04-12 17:24:43.046 INFO  [Inventory-PS11:57] starting Inventory-PS11 mode.
…
2021-04-12 17:25:37.331 INFO  [Inventory-PS11:89] getNext( PS: 11, PoolLocId: 204 )
2021-04-12 17:25:37.331 DEBUG [DBInventory:57] at.syslog.osr.db.Inventory.getNextTray()
2021-04-12 17:25:37.366 DEBUG [Tray:473] Tray(2084523287).addTypeUniqueTask(11, INVENTORY(0, 0, 2084523287, ))
…
2021-04-12 17:25:37.381 INFO  [TowerMF-1:676] Pumping: ejectToZone(2084523287, zone: POOL_1800, psId: 11, orderI
…
2021-04-12 17:25:45.048 INFO  [PickStation-11:2063] Pumping: announceInventoryTray(2084523287)
```

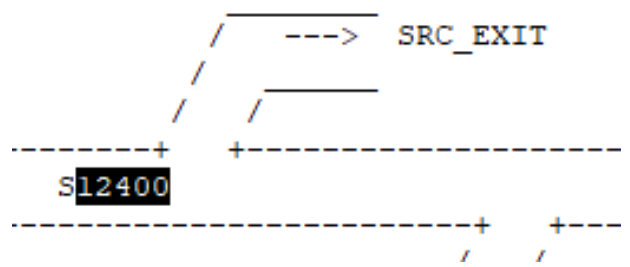
Merge order

EXAMPLE

```
2021-04-19 11:08:53.024 INFO  [PickStation-104] Exec: modeChanged(false,false,false,false,false,true,false,false
2021-04-19 11:08:53.024 INFO  [Merge] starting Merge mode.
…
2021-04-19 11:08:59.128 INFO  [Merge] get next MergeOrder
2021-04-19 11:08:59.128 DEBUG [MergeOrder] getNext( ps:104, TrayType.FULL_TRAY )
…
2021-04-19 11:08:59.540 INFO  [Merge] Found: MergeOrder( id:13041, trayList:  10006719 10019384  )
2021-04-19 11:08:59.540 INFO  [PickStation-104] Pumping: announceMergeOrder( 13041)
2021-04-19 11:08:59.540 DEBUG [Tray] Tray(10006719).addUniqueTask(104, MERGE(13041, 13099, 10006719, ))
2021-04-19 11:08:59.543 INFO  [TowerMF-2] Pumping: ejectToZone(10006719, zone: POOL, psId: 104, orderId: 13041,
…
2021-04-19 11:08:59.543 DEBUG [Tray] Tray(10019384).addUniqueTask(104, MERGE(13041, 13099, 10019384, ))
2021-04-19 11:08:59.544 INFO  [TowerMF-4] Pumping: ejectToZone(10019384, zone: POOL, psId: 104, orderId: 13041,
```

Transport order

EXAMPLE




```
2021-04-19 11:08:40.637 INFO  [TransportTaskExecutor] Exec: addTask( cid=10012615, target=91, orderId=2040772,
…
2021-04-19 11:09:28.874 INFO  [Switch-12400] Exec: containerProcessed(10012615,U=false,R=false,H=false,F=false,
…
2021-04-19 11:09:28.874 INFO  [Switch-12400] Tray(10012615 FULL_TRAY ToProcess height:null, weight:null,
  homeZone:1, sSeg:0, floor:0, sTwr:3, dTwr:null, cTwr:3, lastSeen:12400) :Tasks( TransportTask( sys=SPIDER,
  target=91, orderId=2040772, lineId=2040773 ),  )
2021-04-19 11:09:28.874 INFO  [Tray] Going to finish TransportTask( sys=SPIDER, target=91, orderId=2040772, line
2021-04-19 11:09:28.874 DEBUG [TransportTask] finish TransportTask for 10012615
```

Configuration

All examples taken from version 6.3

Towers

DEFINITION

- Towers are steuer's definition for the aisles in the OSR
- For each aisle, there needs to be a tower definition
- What do we need to configure?
  - ID (number) of the aisle
  - The list of storage zones for that aisle
  - The threshold of the fill level in percent
- Code implementation:
  - All towers are added to the `towerMFSequence` list in the `SteuerCfg.Config` instance

Towers

CODE EXAMPLE

```
def getTower(id, fillLevel):
    return OSR.SteuerCfg.TowerMF(
        id = id,
        towerStorageZones = [
            OSR.SteuerCfg.TowerStorageZone("SBLOCK_%d_STORAGE" % id, "", fillLevel)
        ],
        towerFillLevel = fillLevel
    )

…

steuerCfg = OSR.SteuerCfg.Config(
# ============= the stations ==================
    # towerMFSequence of double/twin lift towers
    towerMFSequence = [
        getTower(1, 99.5),
        getTower(2, 99.5),
    ],
…
```

Picking stations

DEFINITION

- All existing picking stations need to be configured in the steuer config file
- For every picking station, several parameters are used:
  - Picking station ID
  - overload value
  - fetch and prefetch value(s)

- Code implementation:
  - Setup of the picking stations happens through helper functions
  - All picking stations are added to the `pickStationSequence` list in the `SteuerCfg.Config` instance

Picking stations

CODE EXAMPLE

```
def createDefaultPickStationConfig(station_id, overload, max_fetch, max_prefetch, max_fetch_merge, max_prefetch_
    return OSR.SteuerCfg.PickStation(
        id = station_id,
        # The maximum number of trays to be fetched/prefetched for the PickStation
        pickFetchConfig = OSR.SteuerCfg.FetchConfig(max_fetch, max_prefetch, OSR.SteuerCfg.DYNAMIC_FETCH),
        goodsInFetchConfig = OSR.SteuerCfg.FetchConfig(max_fetch, max_prefetch, OSR.SteuerCfg.FETCH),
        goodsAddFetchConfig = OSR.SteuerCfg.FetchConfig(max_fetch, max_prefetch, OSR.SteuerCfg.FETCH),
        inventoryFetchConfig = OSR.SteuerCfg.FetchConfig(max_fetch, max_prefetch, OSR.SteuerCfg.FETCH),
…
    )

                                    #psId, overload, fetch, prefetch, fetch_merge, prefetch_merge
ps1 = createDefaultPickStationConfig(1101,  5,       6,     12,       10,          20)
ps2 = createDefaultPickStationConfig(1102,  5,       6,     12,       10,          20)
ps3 = createDefaultPickStationConfig(1103,  5,       6,     12,       10,          20)

steuerCfg = OSR.SteuerCfg.Config(
# ============= the stations ==================
…
    pickStationSequence = [ps1, ps2, ps3],
…
```

Loop switches

DEFINITION

- All PLC stations on the source conveyor system in the prezone need to be defined as loop switches
- Information required for the configuration:
  - PLC station number
  - Information on where to route in case of errors
  - Some further technical details like whether the station has a height check implemented etc.

- Code implementation:
  - Setup of the loop switches happens through helper functions
  - All stations are added to the `loopSwitchSequence` list in the `SteuerCfg.Config` instance

Loop switches

CODE EXAMPLE

```
def getLoopSwitchConfig(stationId, readingError, unknown, heightCheck, weightCheck, roundRobin):
    return OSR.SteuerCfg.LoopSwitch(
        id = stationId,
        readingErrorRouteIndex = readingError,
        unknownContainerRouteIndex = unknown,
…
        hasHeightCheck = heightCheck,
        checkTrayWeight = weightCheck,
…
)


steuerCfg = OSR.SteuerCfg.Config(
# ============= the stations ==================
…
    # the switches
    loopSwitchSequence = [
                        #stationId, readingError, unknown, heightCheck, weightCheck, roundRobin
        getLoopSwitchConfig(12400,     0,            1,       0,           0,           0),
        getLoopSwitchConfig(12500,     0,            0,       1,           0,           0),
        getLoopSwitchConfig(12501,     0,            0,       1,           0,           0),
    ],
…
```

Diverters

DEFINITION

- Like loop switches, also the PLC stations on the target conveyor system need to be configured – these are called diverters
- Information required for the configuration:
  - PLC station number
  - Information on where to route in case of errors
  - Flag on whether or not to clamp a container when no picking station is in pick mode

- Code implementation:
  - Historically, there are several Diverter classes in the steuer config, but now only GenericDiverter is used
  - All stations are added to the `genericDiverterSequence` list in the `SteuerCfg.Config` instance
  - Also, the station numbers are all added to the `beltDiverterSequence`

Diverters

CODE EXAMPLE

```
def createGenericDiverter(station, clamp, read, unk):
    return OSR.SteuerCfg.GenericDiverter(
        id = station,
        overload = 1,
        clampOnNoPickStationInPickMode = clamp,
        readingErrorRouteIndex = read,
        unknownContainerRouteIndex = unk,
…
        pickStationSelectionStrategies = [any.to_any(OSR.SteuerCfg.SaturationSortPickStationSelectionStrategyCo
    )

…
steuerCfg = OSR.SteuerCfg.Config(
# ============= the stations ==================
…

    genericDiverterSequence = [
                            #(station, clamp, read, unk)
        createGenericDiverter(11702,   1,     0,    0),
        createGenericDiverter(11700,   0,     0,    0),
        createGenericDiverter(11701,   0,     0,    0)
    ],
…
```

Layout

DEFINITION

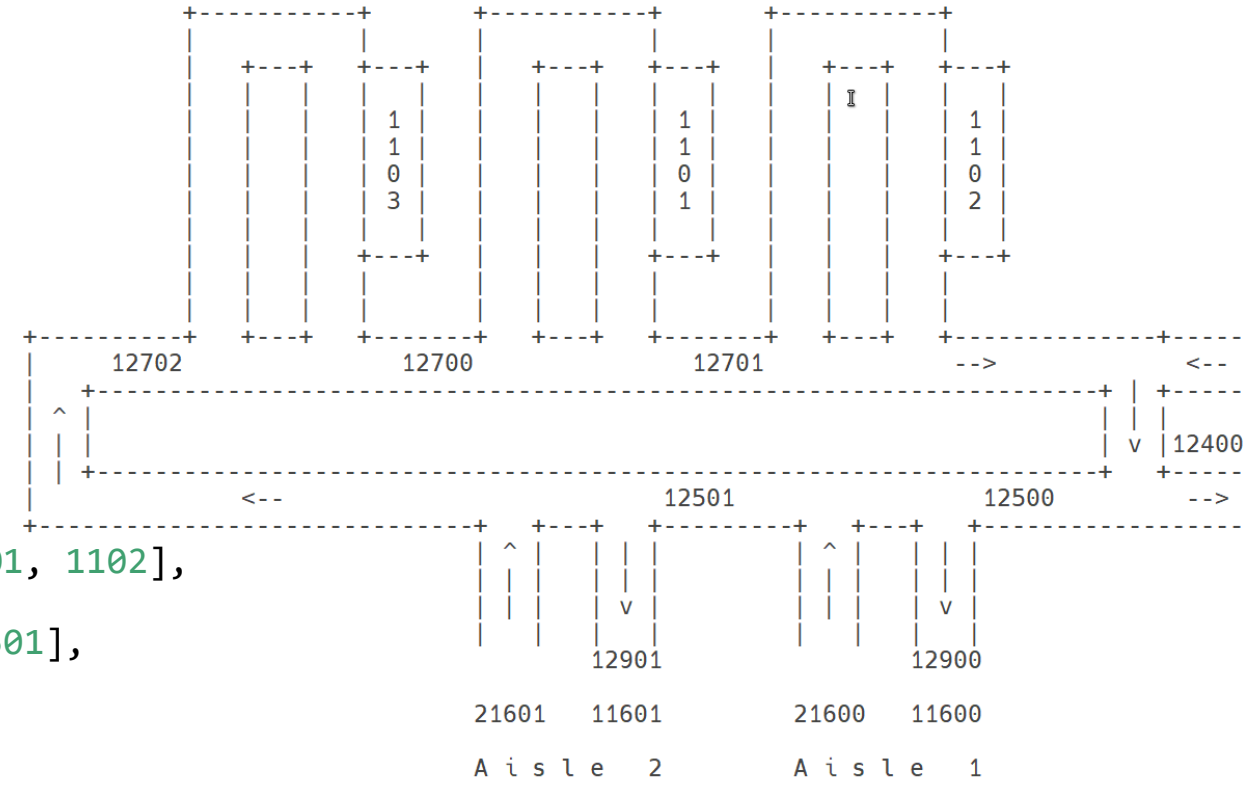
- The layout of all conveyor systems and the components they contain is defined in the layout section of the configuration
- The structural hierarchy is organized in “Floor” and “Segment” instances:
  - For each conveyor system, there is a Floor entry – in a simple setup, there is one Floor
  - Each Floor has at least 1 Segment; if segmentation is used, further entries will exist

- Code implementation:
  - `layout` is structured as a list in the `SteuerCfg.Config` instance
  - This list contains all required instances of the `SteuerCfg.Floor` which in turn has most of its arguments in the `SteuerCfg.Segment` instances

Layout

CODE EXAMPLE

```
layout = [
    OSR.SteuerCfg.Floor(
        id = 0,
        name = "POOL",
        segments = [
            OSR.SteuerCfg.Segment(
                id = 0,
                zoneName = "",
                towerRefSequence = [1, 2],
                pickStationRefSequence = [1103, 1101, 1102],
                fillReaderRefSequence = [12400],
                loopSwitchRefSequence = [12500, 12501],
                scaleStationRefSequence = [],
                fetchStationSequence = [],
                errorStationSequence = [
                    OSR.SteuerCfg.ErrorStationStruct(
                        id = 1101,
                        errorTrayTargetLocation = "",
                        errorTypes = [OSR.SteuerCfg.ALL]
                    )
                ]
…
```



Routes

DEFINITION

- Since steuer is responsible for the flow of containers within the prezone, it needs to have a routing table for all possible stations:
  - PLC stations
  - OSR aisles
  - Picking stations
  - Virtual stations, like SRC_ENTRANCE

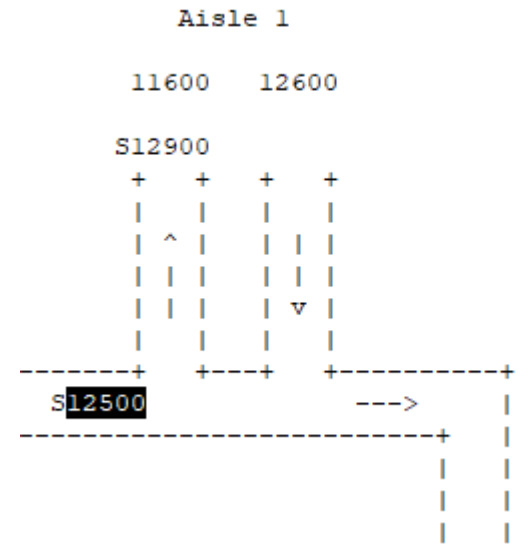
- Code implementation:
  - For all stations, `SteuerCfg.Routes` instances are configured which define what can be reached from the station when going in a specific direction
  - All route entries are added to the `loopLayout` list in the `SteuerCfg.Config` instance

Routes

CODE EXAMPLE

In a route entry, we define:
- direction
- target(s) that can be reached
- station to which we pre-announce the container

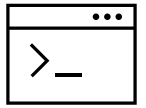
The targets can be a list, but we can also use a shortcut notation there for specific target groups, so “T” means all towers, “P” all picking stations, “F” all fill readers.
“FP103:0” means all fill readers and picking stations in floor 103 (see ID of Floor in “layout”) and Segment 0




```
steuerCfg = OSR.SteuerCfg.Config(
# ============= the stations ==================
…
    loopLayout = [
        # =========== POOL_1000 ========================

        OSR.SteuerCfg.Routes( 12500, [
            OSR.SteuerCfg.RouteEntry(  0, ["FP0:0"], 0),
            OSR.SteuerCfg.RouteEntry( -1, ["1"], 0)
        ]),
…
```

steuer commands and tools

steuer commands and tools



COMMANDS

- `steuer_status` (alias: *`st`*) displays information about all containers it has to handle, as well as the announcement lists per picking station

Tips & Tricks

Tips & tricks

Tips

- Use the IDL file (`/kisoft/user/osr/source/src/lager/idl/steuer_cfg.idl`)
  - Shows all valid arguments and options for the steuer configuration, including some descriptions
- Explore the template files (`/kisoft/user/osr/source/src/lager/templates/`)


KNAPP
making complexity simple

KNAPP

Steuer

Handout

BURGSTALLER Martin

TODO: UH000575-b

TODO: ID4049/02

01/2019

Table of contents

1. Standard configuration

To configure the steuer you need to know about all the connections the steuer has to other processes.

It connects directly to the picking stations and manages the “fetch” and “prefetch” to those, depending on the order start configuration either the steuer directly starts orders to the picking stations, or it just manages the work packages sent by “knoedl” (our database order start).

There also needs to be a connection to the hio command process to get information about new picking orders and their target containers, if the steuer starts the orders.

And the third important connection is the spider connection. On one hand the steuer can request containers from the OSR for the picking stations and on the other hand spider needs to be able to tell steuer where to bring containers to, if they should go to a specific location which is only reachable via the prezone loop.

1.1. Pick stations

At the upper most part of the steuer config the function for the “PickStation” creation can be found. In there we have got several parameters to configure globally and three which are handed over to the function and are unique, more or less.

The first variable we hand over to the function are the pick station id, which should be the same throughout the whole system, normally they start with 21. The other two are the fetch and the prefetch values.

The fetch defines how many containers are allowed to be on their way to the specific picking station or can fit onto the pick finger, including the ones in the source bays. When configuring the fetch just count the jam places on the pick finger and if the normal fetch is used subtract 1. That’s because there could always be a container coming to the station from another pick station.

The prefetch value is the number of containers which can be assigned to the spider, including the ones which should get fetched. The difference between the fetch and prefetch value will be announced to the spider as prefetch jobs. Those will be buffered on the exit lifts buffers. When configuring the prefetch take the sum of your exit buffers divide it by the number of pick stations and then add the fetch value. If you do so you will get a rough value to work with, which can get fine-tuned afterwards.

The fetch and prefetch values will be added to the fetch configs for each individual order type, as well as the fetch type, from SRC 6.1 onwards. Older SRC versions do have a specific config value where you can select the fetch mode globally.

At the current state you can select from those fetch modes: FETCH, DYNAMIC_FETCH, STRICT_DYNAMIC_FETCH, NO_FETCH, BATCH_FETCH. Where the FETCH is the classic mode which always fills up to the configured fetch value. The DYNAMIC_FETCH will fetch by orders, meaning it will start to fetch the next order only if the previous one is fetched, but it can fetch as many orders simultaneously as there are target bays in the pick station. A stricter version is the

STRICT_DYNAMIC_FETCH. This mode is only allowed the fetch one order after the other and requires the first order to be fully fetched, before starting to fetch the second one. Next is the NO_FETCH, which is only to be used in combination with the “knoedl” and dispatch fetch from spider, but that’s a very very special case. Finally, we’ve got the BATCH_FETCH which fetches containers according to their batch, also a special case and not all too often used.

The next parameter to be configured is the “softLimitManualOrderLines”, which is only used if the target containers are married directly at the picking stations and therefore manual orders are used.

If the host sends the target containers and steuer has to divert them to the pick stations you need to use the overload value, which tells the steuer how many containers fit between the diverter and the target bays.

Next is “useManualBatch”. This defines if the steuer should announce manual orders in batches, only useable if there actually are batches created at the order splitting.

The “mixerChannelSequence” describes the target bays of the picking stations and how the steuer should announce sources to the pick station. The maximum number of channels is the number of target bays and for optimal usage of the fetch it should always be at the maximum. There are different slot types for the different order processing modes. The most common one is the TPC_SLOT, which needs to be used for normal steuer picking.

“emptyOutFillerOnPickModeFinish” declares if the target containers should leave the pick station, if the pick mode is stopped, or if they should stay there and therefore need to be picked at the current station. In case the filler is emptied, the picking orders will be cancelled.

In case that the “knoedl” is used to assign and start orders to the pick stations the parameter “workPackageHandling” must be set to 1.

The last parameter will only be found from SRC 6.1 onwards and is the “psSequencerConfig”. That’s used for the dynamic fetch to define if there is a mini loop connected to the pick station, which can be used for sorting the orders. If so, the dynamic fetch can fetch more totes at once.

1.2. Towers

There are two different tower sequences. Those are for different types of OSR. The first one is the normal “towerSequence”, where the towers are configured, those need to be used for OSR 15 systems and only have an id and a filling degree to be set up.

The other one is used for all the rest of OSR systems and is called the “towerMFSequence”. Each tower there needs to be a “towerMF”. The difference in configuration is that the MF towers can handle storage zones, where the storage zones can have a home zone defined. The standard configuration would be that each MF tower would only have the “PRODUCT_STORAGE_AISLE” zone of the current aisle defined. But for some systems where an aisle could have multiple zones inside all those zones need to be configured for the tower. It could also happen that there are special storage zones which are only reachable by one specific other zone, then the home zone

needs to be configured. That only applies if the steuer can reach the home zone and is not in charge after bringing the container there.

The fill levels should always stay at the default value, which is “99.5” unless there is a good reason for changing it.

1.3. Loop Switches

The loop switches are all the plc stations controlled by steuer on the prezone-loop for the source containers. All of them need to be configured in the “loopSwitchSequence” first before they can be used elsewhere in the config.

Each loop switch has an id, which is the station number from plc, and a route index for reading errors and one for unknown containers, those should be configured after the route configuration. Then there is the “removeTrayOnDiversionToExit” value, which is usable for stations with a direction to an exit location. The “cancelTransportTaskOnDiversionToWrongPosition” tells the steuer if it is necessary to cancel transport tasks which have been transmitted by spider when the container is not diverted to the position we wanted it to go to. Might be needed if we can not come back from there, or if we are using an automated empty container lane, which could get full without any of our processes knowing it. The “hasHeightCheck” and “checkTaraWeight” can be configured if the station has either a height control on plc side, or if there is a scale which should check the weight of the containers. And if the weight check fails you can use the “uncheckedTrayWeightRouteIndex” to define where those unchecked containers should be diverted to.

The “containerMovementConfigs” is a bit tricky and not often used. Its for creating transport orders for containers which are diverted to a specific target and also get booked to a new location. E.g:

```
OSR.SteuerCfg.ContainerMovementStruct(
   targetId = 22,
   transferLocation = "TO_OFFLINE_POOL",
   movementTargets = ["OFFLINE_TKS", "CIRCULATION"]
)
```

In that example all containers which get diverted to the direction of station 22, which is a pick station, will be booked to the location "TO_OFFLINE_POOL" and then a transport order will be created. The transport order will have as many lines as there are “movementTargets” configured and they will be created in exactly the order the targets are in the list. Means for the example the transport order will bring the container first to "OFFLINE_TKS" and after that to "CIRCULATION".

The last two parameters are also not often used. They are “restrictedTowerSelectorRange”, which was developed for HugoBoss and tells the steuer how many other aisles to select before the same one can be selected again, and “roundRobinPossibleRoutes”. The round robin option will only work if there are multiple directions configured for the same target.

1.4. Floors

The floors describe the prezone-pools you have. For each individual pool there needs to be a floor configured in steuer.

A floor contains an id, a name, a list of segments and how many segments are reachable from that floor. The id can be freely selected, but in standard the first one starts with 0 and is increasing from there on.

The could be anything you like, but regarding the readability and standards the name of the connected pool location should be used. The steuer will use the floor name to fetch containers, which means that the spider needs to know the names as well and needs to have routes to the floor.

Normally each floor has only one segment. Only some bigger special projects will use segments in the prezone. The individual segments also have an id which starts at 0 and should be increased by 1 for every additional segment of the current floor. If you have multiple floors each floor can have a segment with the id 0.

The segments “zoneName” only needs to be used if there are multiple segments. In that case the zone name might also be a home zone for one of the towers. But the reachable towers must be configured in any case with the “towerRefSequence”. In there the towers must be in the same order the containers will reach the towers when they drive around on the loop.

The pick stations need to be added in the correct order to the “pickStationRefSequence”. Same goes for the fill readers which must be added to the “fillReaderRefSequence” and loop switches to the “loopSwitchRefSequence”. Where the first fill reader in order is the one for the first aisle in sequence and the first loop switch is the first one of those after the first fill reader.

For explanation: A fill reader is a loop switch which decides which aisle a container should go to. At a standard OSR there will be only one fill reader per segment of a floor and that’s basically the first loop switch on the loop. Means the fill reader is configured like the rest of the loop switches as explained in the previous chapter.

After the loops switch sequence there are two special sequences which won’t be used very often. Those are the “scaleStationRefSequence” and the “fetchStationSequence”.

The scale station sequence obviously should contain all scale stations in that segment, but as explained earlier they are not very common to be used. Same goes for the fetch station sequence. The fetch stations are only used at systems where the dynamic or strict dynamic fetch are active and tell the steuer at which station a container can be considered as fetched. So basically, it’s the point of no return for the containers, where they will get to the pick stations no matter what.

The last part of the segment is the error station sequence where all error stations are configured. Normally you would just select a suitable pick station and send there all containers with any kind of error. But you can also split up the error stations by type of error, or you could use an exit location as an error station.

1.5. Diverters

The diverters are kind of like the loop switches but are controlling the target container area instead of the source area. And there are several different types of diverters where each one of the types does its job a little bit different. The most basic one is the “TogglingDiverter”, which just announces target containers and therefore orders to the next free pick station. Then there are the other types with the “FillingDiverter”, “SpreadingDiverter”, “SplittingDiverter”, ”SplitToggleDiverter”, “RobotDiverter”, “WorkPackageDiverter” and the “GenericDiverter”. Each of those types generally just does what its name says and most of them are very rarely used.

The newest kind of diverters is the “GenericDiverter”, which can be configured further. There can be several different features added to it, so that it basically can replace any of the other diverters and combine some of their functions.

Whatever diverter you want to configure needs to be added to the correct diverter sequence in the config. And all of them together need to be added to the “beltDiverterSequence” in their order on the loop, but there only their ids need to be added.

1.6. Routes

For each station which is controlled by steuer you need to configure the possible routes. At every station steuer will check the routes for the containers target and will announce the configured position to plc. A route entry could look like that:

```
OSR.SteuerCfg.Routes( 12500,[
    OSR.SteuerCfg.RouteEntry(  0, ["2", "3", "4", "92"], 0),
    OSR.SteuerCfg.RouteEntry( -1, ["1"],  1)
]),
```

This example is the route entry for a fill reader, which is, as you should know by now, a loop switch. Which station the route is for is defined in the first line, in the example it’s the station 12500. The following lines are the route entries. The first value in each entry is the divert direction from plc, normally 0 is stay on loop and -1 is divert from loop. The values in the list are the reachable stations. So, if a container is clamped at 12500 and needs to go to aisle 3 steuer will send the container to position 0, if the container should go to aisle 1 instead steuer would need to divert it to -1. The last value in each of the route entries is the station which should be announced to, where 0 means no station should get a preannounce.

If the target is not listed in the route entries of a station steuer will send the container to the default route index, which normally is 0, so the first configured route entry.

If there are multiple floors its possible to configure routes where the target isn’t a plc direction, but a floor id. Those routes have to be configured for the aisle stations and could look like that:

```
OSR.SteuerCfg.Routes( 1,[
    OSR.SteuerCfg.RouteEntry(  10, ["91","92"], 0),
    OSR.SteuerCfg.RouteEntry(  11, ["100"], 0),
    OSR.SteuerCfg.RouteEntry(  21, ["200"], 0),
    ]),
```

In this example the station is aisle 1 and the target positions are the floors 10, 11 and 21. Now if the steuer wants to get a container to station 100 it needs to be brought to aisle 1 and from there spider can bring it to floor 11. But for the spider to be able to bring it there steuer must tell spider, and that's what the route entry is for.

1.7. Other parameters

In the other values part of the config are some special parameters which need to be set globally like the "storageSystemTransporterId". That's the steuer's global transporter id which is used by spider and should normally be 0.

The "alternativeTowerSelection" is only used for multi segment systems as well as the "crossSegmentTrafficAllowed". When the alternative tower selection is enabled steuer can send a container to a different segment, if it passed by the originally selected tower. That can only happen if the cross-segment traffic is allowed.

The "towerPollingTime" is the timeout for connecting to the spider and the "stationPollingTime" is the same for the plc stations, so basically the timeout for plc adapter connection.

The "overweightLimit" defines how much weight a container is allowed to have, before it must leave the system or go to an error station.

The "coreThreadPoolSize" is not used at all, just leave it alone. Same for the "cancelTransportTaskModePropertyName", just leave it as is.

With the "preannounce" you can define if steuer should preannounce from one station to the other, that's not really used anymore, because our servers and the plc is fast enough now. At newer sites it could cause problems if turned on.

The "trayBasedOrderAssignment" is only used for multi segment systems and manual orders, where the pick station is selected based on the source tower of the selected tray.

If you want to ignore the reading of the entrance station on the target container loop you can set the "ignoreEntranceReader" value to 1. Might be usable if it is a virtual one.

Also, for segmented systems is the parameter "isolateZonesForPicking". But this one is used if there are transport containers involved, they then will be sent to a suitable pick station. For this to work the splitting diverter must be used.

For customers, where an order priority is sent, the "forceTopPriorityFirst" should be activated. With this variable active the steuer will sort pick orders depending on their priority. But this might not work, if we get the target containers from WCS and the target containers are not able to loop within the stauer boundary.

The "testMode" could be used for mass testing with inventory orders. But its old and might not be suitable anymore.

Finally, the "taskPriority" is basically the same as the "forceTopPriorityFirst", but for work packages sent by "knoedl".

2. Process startup

For the steuer to be able to start there are serval stations and listeners which need to exist in naming. First is the listener from spider to steuer. That's why the spider always starts before the steuer.

The next one is a bit trickier. And that's the pick station listeners. The listeners themselves are getting set by steuer, but the stations must exist to be able to set the listener. The stations only exist if the pick stations started up to a certain point. And its mandatory for the steuer to set all pick station listeners.

On the other hand, its mandatory for the opas that the listeners to them are set, otherwise they won't boot up fully.

Once all pick station listeners are set and the steuer can boot up it will connect to the virtual entrance and exit stations, those are set by hio command. After that the connection to the towers is established to get information about the reachability and filling degree.

After that the steuer is ready to rock, even if the route configuration might not be a 100% correct, so be careful.

3. Good practices

One thing which should be done pretty early in the project phase is to set the fetch and prefetch values correctly, or at least as described in the chapter about fetch and prefetch. That can be done as soon as you are on site and help to find out about possible problems on the loop and with the performance. The same applies for the overload of the pick stations if the steuer is order start.

Another good practice would be to do the routing as detailed as possible and just use one route entry per direction. Make it easy for the steuer to know where it has to send containers. To make it easier you can use the shortcuts for all pick stations, all fill readers, or all towers on the floor. Just use “P” for the pick stations, “F” for the fill readers and “T” for the towers and then add the floor and segment id, like so “PFT0:0”. That would mean all of them at floor 0 in segment 0.

The next to announce for the route entries needs to be used for aisle entrances and for routes to exit locations at the station which should book the container onto the exit.

Check the entrance and exit locations, there must be one entry for each of them. At least there should be the stations 90 and 91, which are the “SRC_ENTRANCE” and “SRC_EXIT”. The also have to be configured in the database at the transporter locations for the steuer transporter and the x-coordinate must be the station number in the steuer config.

4. Common mistakes/errors

If the steuer won't start check if the opas are starting up. If the opa processes are not ready to be used and started yet, but you need the steuer, then simply remove them from the pick station sequence. The rest of the pick station config in steuer can stay in place.

Always check if the station 90 and 91 are created by hio command otherwise steuer can't start as well.

There must be routes configured for the station 90 and to the station 91, otherwise you will get a config error prompt in the steuer log.

Status	Finished
Started	Tuesday, 1 July 2025, 2:53 PM
Completed	Tuesday, 1 July 2025, 3:01 PM
Duration	8 mins 26 secs
Marks	7.27/10.00
Grade	**72.67** out of 100.00

Question 1

Partially correct

Mark 0.50 out of 1.00

Which statements below regarding segmented systems are correct? Check all that apply:

- [x] a. Source trays stay in their segment ✓
- [x] b. Source trays travel a shorter distance ✓
- [x] c. In order to fulfill an order at any picking station, mirrored stock is required in the OSR ✓
- [ ] d. Target containers stay in their segment
- [ ] e. The target is to increase performance
- [ ] f. Segmentation is only possible in orderstart systems
- [x] g. Segmentation is only possible when having at least 2 OSRs ✗
- [ ] h. Each segment may have 3 picking stations maximum

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: Source trays stay in their segment, The target is to increase performance, Source trays travel a shorter distance,

In order to fulfill an order at any picking station, mirrored stock is required in the OSR

Question 2

Correct

Mark 1.00 out of 1.00

At which point must the decision be taken which aisle in the OSR a container will be stored to?

- ○ a. At station 12501
- ○ b. At the picking station
- ○ c. At the fetch station
- ◉ d. At the fill reader ✓

Your answer is correct.

The correct answer is:
At the fill reader

Question 3

Partially correct

Mark 0.50 out of 1.00

Which parameters control the number of source trays assigned to the picking stations?

- [x] a. fetch
- [ ] b. prefetch
- [ ] c. overload
- [ ] d. routing_table

Your answer is partially correct.

You have correctly selected 1.

The correct answers are:
fetch,

prefetch

Question 4

Incorrect

Mark 0.00 out of 1.00

What has been implemented in steuer to optimize the fetching of source trays?

- a. transactional fetching
- b. redundant fetching
- c. optimized fetching
- d. dynamic fetching
- e. randomized fetching

Your answer is incorrect.

The correct answer is:
dynamic fetching

Question 5

Correct

Mark 1.00 out of 1.00

Which parameter controls the number of target containers that will be assigned to picking stations?

Answer: overload

The correct answer is: overload

Question 6

Correct

Mark 1.00 out of 1.00

Which statements about the steuer controlled prezone are true?
Check all that apply.

- [ ] a. Only the source loop is connected to the picking stations
- [x] b. The source loop is usually located on top of the target conveyor
- [x] c. PLC stations on the target conveyor are called diverters
- [x] d. PLC stations on the source conveyor are called loop switches
- [ ] e. A fill reader is the first reading station after the aisles
- [ ] f. Target containers always come from the WCS area

Your answer is correct.

The correct answers are:
The source loop is usually located on top of the target conveyor,

PLC stations on the target conveyor are called diverters,

PLC stations on the source conveyor are called loop switches

Question 7

Correct

Mark 1.00 out of 1.00

Which helper command shows an overview of the steuer environment?

Answer: st

Yes, this is a shortcut for "steuer_status"

The correct answer is: steuer_status

Question 8

Partially correct

Mark 0.27 out of 1.00

What does the below excerpt from a steuer log file tell you? Check all options that apply.

```
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ------------ PS(103) --------->>>
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10018087, PICK(412255, 1257670, 10018087, 2332596) }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10001892, PICK(412270, 1257723, 10001892, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList -- : { 10027101, PICK(412270, 1257717, 10027101, 2332692), fetched }
2021-04-19 16:24:01.153 DEBUG [PickStation-103] trayList: ----------------------------<<< size: 3
```

- [x] a. The target containers' numbers are 7 digits long and start with "233" ✓
- [ ] b. Pick order 412270 requires two different source trays
- [ ] c. Picking station 103 is currently turned off
- [x] d. The log level for the steuer process is in debug mode ✓
- [ ] e. Picking station 103 has 2 pending pick orders
- [x] f. The order numbers of the pick orders have 8 digits and start with "100" ✗
- [ ] g. There are currently 3 source containers in the bays of the picking station
- [x] h. Two source trays have been ejected to the prezone, one source tray is still in the OSR ✓

Your answer is partially correct.

You have correctly selected 3.

The correct answers are:
Picking station 103 has 2 pending pick orders,

Pick order 412270 requires two different source trays,

Two source trays have been ejected to the prezone, one source tray is still in the OSR,

The target containers' numbers are 7 digits long and start with "233",

The log level for the steuer process is in debug mode

Question 9

Correct

Mark 1.00 out of 1.00

Which statements concerning the code below are correct? Check all that apply:

```
OSR.SteuerCfg.Routes(12500, [
     OSR.SteuerCfg.RouteEntry(0, ["P0:0", "92"], 12501),
     OSR.SteuerCfg.RouteEntry(-1, ["1"],  1)
]),
```

- [x] a. On the default route, an additional SRC exit can be reached
- [x] b. When going "straight", there will be a pre-announcement to station 12501
- [x] c. On the default route, all picking stations can be reached
- [ ] d. Station 12500 is a diversion to a picking station
- [ ] e. PLC station 12500 is on the target conveyor line
- [ ] f. Route entries are only required for entrances into aisles
- [x] g. The PLC station is on the "default" floor and segment (0)
- [x] h. Station 12500 is an entrance to aisle 1

Your answer is correct.

The correct answers are:
Station 12500 is an entrance to aisle 1,

When going "straight", there will be a pre-announcement to station 12501,

On the default route, all picking stations can be reached,

On the default route, an additional SRC exit can be reached,

The PLC station is on the "default" floor and segment (0)

Question 10

Correct

Mark 1.00 out of 1.00

Which statements regarding the pre-announcement of a container to a PLC station are correct? Check all that apply:

- [ ] a. Pre-announcement can only be done on PLC stations on the target conveyors
- [x] b. Pre-annoucement is supposed to increase performance
- [ ] c. There will be no more "announceContainer" message in the logs
- [x] d. There will be no more "containerClamped" message in the logs
- [ ] e. There will be no more "containerProcessed" message in the logs

Your answer is correct.

The correct answers are:
There will be no more "containerClamped" message in the logs,

Pre-annoucement is supposed to increase performance