Univerza *v Ljubljani*
Fakulteta *za strojništvo*

University of Ljubljana

Faculty of Mechanical Engineering

Tomaž Požrl, Marko Corn

Industrial IoT through JavaScript language and Node.js run-time environment

Project materials for 2018 LDSE Summer School on Industrial IoT

Ljubljana, 2018

Content

List of Pictures v

List of Tables vii

1. Introduction 1

2. Raspberry Pi platform and Node.js environment 5
2.1 Operating system . . . 5
2.2 Nodejs . . . 6
2.2.1 Installation . . . 6
2.2.2 Hello world example . . . 7
2.2.3 Web API . . . 8
2.2.4 Services . . . 10
2.2.5 Communication between nodes . . . 10

3. Beckhoff PLC platform and TwinCAT environment 13
3.1 Programmable logic controllers (PLC) . . . 13
3.2 Staudinger warehouse model . . . 14
3.3 Beckhoff automation and control systems . . . 17
3.3.1 Embedded PC CX 8090 . . . 18
3.3.2 I/O devices . . . 19
3.4 TwinCAT . . . 21
3.4.1 TwinCAT Usage . . . 21
3.4.2 System Manager . . . 22
3.4.3 PLC Control . . . 26
3.5 Structured Text (ST) . . . 33
3.6 Modbus TCP . . . 34
3.6.1 Modbus TCP in Beckhoff systems . . . 35

A 2018 Summer School Project Assignments 37
A.1 Module 1: Smart Package Node . . . 38
A.2 Module 2: Retailer Node . . . 40
A.3 Module 3: Warehouse Node . . . 41
A.4 Module 4: Transportation Node . . . 52
A.5 Integration . . . 53

References **55**

List of Pictures

1.1 Traditional supply chain system 1
1.2 Decentralized supply chain system 2
2.1 Raspberry Pi computer 5
2.2 Output of the browser 8
2.3 Response of web server in json format 9
3.1 Staudinger Warehouse Model 226001 15
3.2 Staudinger Warehouse Model 226001 System Layout 17
3.3 Beckhoff Embedded PC CX 8090 19
3.4 TwinCAT System Manager - open new file 23
3.5 TwinCAT System Manager - System Configuration - Choose Target . 23
3.6 TwinCAT System Manager - PLC Configuration - Append PLC Project 25
3.7 TwinCAT System Manager - I/O Configuration - Set TwinCAT to Config mode 25
3.8 TwinCAT System Manager - I/O Configuration - Check Configuration 26
3.9 TwinCAT System Manager - I/O Configuration - Activate Configuration 27
3.10 TwinCAT PLC Control Overview 27
3.11 TwinCAT PLC Control - Global variables 29
3.12 TwinCAT PLC Control - Local variables 30
3.13 TwinCAT PLC Control - Library Manager 31
3.14 TwinCAT PLC Control - Main Programming Window 31
A.1 MCS of distributed supply chain system 37
A.2 Example of a Simple Warehouse GUI 51
A.3 Example of process of delivering package to the customer 53

List of Tables

3.1 Staudinger Warehouse Model 226001 Technical Data 15
3.2 Staudinger Warehouse Model 226001 Sensors and Actuators 16
3.3 Beckhoff CX 8090 Technical Data 20
3.4 Modbus Object Types 34
3.5 Modbus Basic Functions 35
3.6 Modbus to TwinCAT PLC Mapping 36
A.1 Warehouse Model Inputs - Mapping from Connected Pins to Program Variables 44
A.2 Warehouse Model Outputs - Mapping from Connected Pins to Program Variables 44
A.3 Warehouse Model Positions Definition 45
A.4 node-modbus Module TCP Connection Settings 49
A.5 node-modbus Module Examples 50

1. Introduction

The main goal of this summer school is to give you an insight into the field of industrial internet of things (IIot). We will be discovering application use of IIOT by building a small decentralized supply chain system (online store) that will be able to deliver purchased products to the costumer front door.

Traditional supply chain systems that are used by big retailers like Amazon rely on centralized systems to take orders, to monitor storing conditions, ordering transports and other activities. In contrast, the decentralized supply chain systems do not have a central system, they have a distributed system that consists of multitude of smart 'mini' systems or nodes that communicate with each other to achieve their goals. In Figure 1.1 we can see a simplified example of a traditional supply chain system.

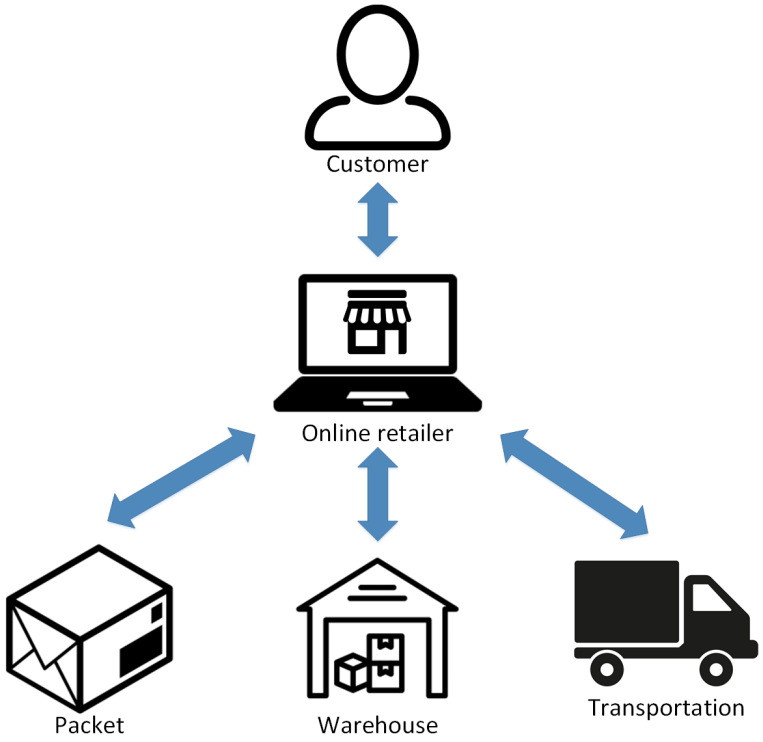

Figure 1.1: Traditional supply chain system

In presented supply chain system the retailer plays the central role in the whole process. It communicates with the customers via online store, it manages the packages that are stored in warehouses and organizes the transportation services.

This type of systems rely on the central control systems to efficiently control theirs supply lines.

Decentralized supply line systems don't have a central control system; they have a distributed system of nodes that communicate with each other directly to achieve their goals. In this type of systems each node is a free agent that maximizes its own efficiency which consequently maximizes whole supply chain system efficiency. In Figure 1.2 an example of a decentralized supply chain system (that has been build from the example of a centralized supply chain system seen in Figure 1.1) is presented.

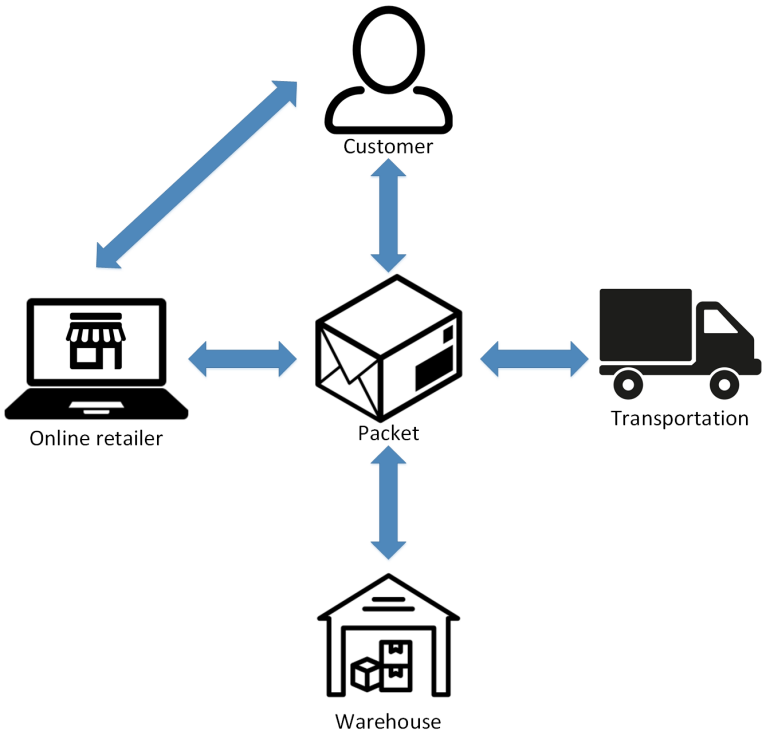

Figure 1.2: Decentralized supply chain system

Distributed supply chain system consists of multitude of nodes that communicate with each other to arrange storage, transportation and other tasks. Our distributed supply chain system defines four types of nodes: smart package, retailer, warehouse and transportation.

- **Smart package node**

  Smart package node represents product that is purchased by the user and has to be delivered to the user home address. Smart package node can also be used to monitor package environmental variables like temperature, humidity, accelerations and others that can be used to monitor storing and transportation conditions.

- **Retailer node**

Retailer node represents an online shop provider. Retailer does not own, store or transport packages it just sells them via its online shop platform. Its main function is to provide shop services and advertisements for the products.

- **Warehouse node**

  Warehouse node represents a storage facility for storing smart packages. Its main functions are storing packages and loading and unloading of packages to the transportation node.

- **Transportation node**

  Transportation node represents a transportation service that is able to deliver packages to desired location. This node can utilize any type of transportation from a ship, an air plane, drones, autonomous vehicles and any other as long as they are able to provide transportation of the packages.

2. Raspberry Pi platform and Node.js environment

Implementation of this project will be based on IoT approaches which in most cases utilize web technologies. This means that each node has to provide a web server through which it can communicate with other nodes via web protocols. In order to host a web server on each node the node must have a proper hardware and software equipment.
The technology has made a huge progress towards cheap mini computers like Raspberry Pi that are more than capable of serving the needs of our application which enables us to use them for our project. We will be using Raspberry Pi 3 Model B+ that is illustrated in Figure 2.1

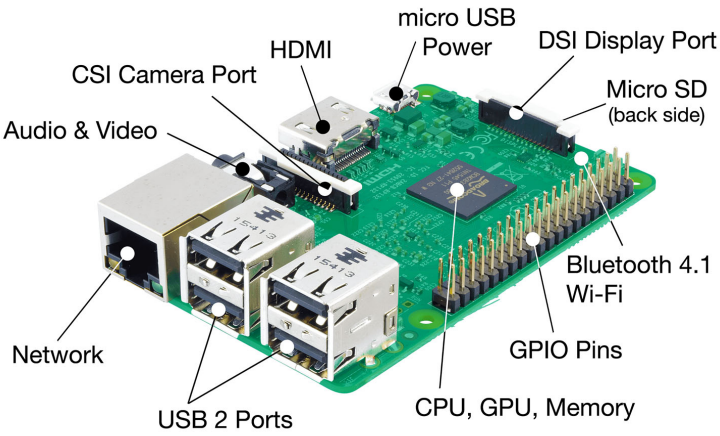

Figure 2.1: Raspberry Pi computer

Raspberry Pi has an integrated Wi-Fi module for wireless connectivity for nodes that change locations like packages and transportation and an Ethernet port for wired connectivity for nodes that don't change locations like warehouses and retailers.

2.1 Operating system

Each node must have an operating system on which the server is running. Nodes will be running a Linux operation system called Raspbian. Installation

instructions for the selected operating system can be found online `www.reeve.com/Documents/Articles\%20Papers/Reeve_RPi_BasicSetup.pdf`. After installation of the operating system we can proceed with the installation of a Node.js web server.

2.2 Nodejs

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world. In this chapter we present how to install Node.js and write three example applications that will help us build our application of decentralized supply chain system.

2.2.1 Installation

Installation of the Node.js server on Linux operating system is done using the terminal console by following next steps:

1. First we update our local package index with the following command. When prompted for password (because of `sudo` command) we type it in.

   ```
   sudo apt-get update
   ```

2. Then we type a `curl` command that utilize a cURL tool to download the latest package from the server where Node.js package is hosted.

   ```
   curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
   ```

3. Then we install Node.js package from the repository.

   ```
   sudo apt-get install nodejs
   ```

4. To compile and install native add-ons from npm you may also need to install build tools.

   ```
   sudo apt-get install -y build-essential
   ```

5. To check if Node.js server is successfully installed on our system in the console type:

   ```
   node -version
   ```

2.2.2 Hello world example

After successful installation of Node.js server we create an example 'Hello World' project and run it as a standalone server on our computer.

1. Prepare folder for your project.
   We open console and type two commands. These commands are universal for whatever OS you'll be running. The first command will create a new directory inside the directory you are currently in. The second will change into this newly created directory.

   ```
   mkdir helloworld
   cd helloworld
   ```

2. Initialize your project and link it to npm.
   After creating your directory, you will need to initialize a project and link it to the Node.js package manager npm. npm is a public repository where all Node.js packages are stored. Packages can be viewed as bundles of code, like modules, which carry out a specific function.

   ```
   npm init
   ```

   This creates a package.json file in your application folder. The file contains references for all npm packages you have downloaded to your project. The command will prompt you to enter a number of things. You can enter your way through all of them.

3. Install Express in the application directory

   ```
   npm install express --save
   ```

   The install command will go ahead and find the package you wish to install and install it to your project. You will now be able to see a *node_ modules* folder gets created in the root of your project. This is a crucial step, as you will be able to require any of the recently installed files in your own application files. The addition of `--save` will save the package to your dependencies list located in the package.json file in your application directory. Express is a fast, unopinionated, minimalist web framework for Node.js. It gives you a set of robust and easy to use tools to get your web application up and running. Express has become so popular it now is the de facto standard in the vast majority of Node.js applications today. It's use is strongly encouraged.

4. Start WebStorm IDE, open the *helloworld* directory as a project and create a file named index.js. Write the following code to the file:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
   res.send('Hello World!');
});
app.listen(3000, function () {
   console.log('Example app listening on port 3000!');
});
```

Here is where you will need to use the package which was recently installed. The first line declares a variable which will contain the module called express, grabbing it from the *node_ modules folder*. The module is actually a function. Assigning the function call to another variable gives you access to a predefined set of tools which will in a great deal make your life much easier. You could view the variable `app` as an object, whose methods you are utilizing to build the actual program. The listen method starts a server and listens on port 3000 for connections. It responds with "Hello World!" for get requests to the root URL (/). For every other path it will respond with a `>404 Not Found`.

5. Run the application by typing the command

   ```
   node index.js
   ```

   After running the command, load http://localhost:3000/ in a browser to see the output (Figure 2.2). You should also see that *'Example app listening on port 3000!'* message gets logged to the command line.

Figure 2.2: Output of the browser

2.2.3 Web API

A Web API is an application programming interface for either a web server or a web browser. It is a web development concept, usually limited to a web application's client-side.

A server-side web API is a programmatic interface consisting of one or more publicly exposed endpoints to a defined request–response message system, typically expressed in JSON or XML, which is exposed via the web — most commonly by means of an HTTP-based web server (in our case Node.js server).

Endpoints are important aspects of interacting with server-side web APIs, as they specify where resources lie that can be accessed by third party software. Usually the access is via a URI to which HTTP requests are posted and from which the response is thus expected. Endpoints need to be static, otherwise the correct functioning of software that interacts with it cannot be guaranteed. If the location of a resource changes (and with it the endpoint) then previously written software will break, as the required resource can no longer be found at the same place. Endpoint example of a live trading activity of cryptocurrency exchange Poloniex: `https://poloniex.com/public?command=returnTicker`.

We have already written an API endpoint in our Hello world application that can be called with `http://localhost:3000/`. Now we will add another endpoint to our project that will accept two numbers and return their product. In our code we add a new `app.get` function as shown below:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.get('/clcSurface', function (req, res) {
let a=req.param('a');
let b=req.param('b');
    res.send({surface:a*b});
});
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

If we call this new endpoint we must include two parameters a an b in the URL. The proper URL is `http://localhost:3000/clcSurface?a=10&b=3` and the response is a json string, as shown in Figure 2.3.

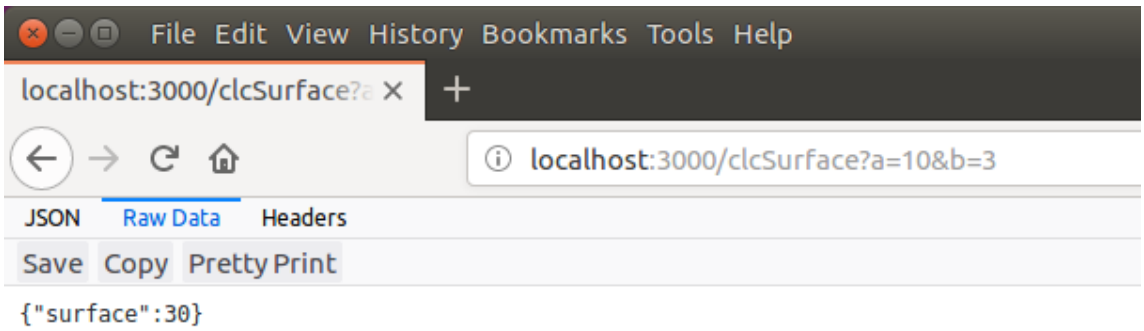

Figure 2.3: Response of web server in json format

2.2.4 Services

Our web server will not be used only for serving http requests. We will be using it also for other purposes that are required for a certain type of node. For an example a package node has to communicate with a GPS module to obtain its location. In order to run this type of services we have to add a periodically repeating function that can communicate with other systems. After the program starts the server (with code line `app.listen...`) we place our code that executes periodically.

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.get('/clcSurface', function (req, res) {
let a=req.param('a');
let b=req.param('b');
    res.send({surface:a*b});
});
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});

let i=0;
setInterval( function() {
i++;
console.log('Counter:'+i);
},1000);
```

To periodically execute code we are calling a `setInterval` function that is a global Node.js function. We set an execution of this function to 1000 ms as can be seen as second parameter to the `setInterval` function. In JavaScript programming language it is possible to pass function as an input argument to another function which we did as first parameter of the `setInterval` function where we increase counter `i` and write to the console state of the counter.

Because Node.js is an asynchronous event-driven system we can place more `setInterval` functions as they functionality is going to be executed when their interval event happens.

2.2.5 Communication between nodes

Each node represents a WEB server with its endpoints that can be evoked via web browser call. We have described a way of calling this endpoints by typing strings into

browser URL filed. Here we will describe a way in which nodes can call each other automatically, which is crucial function for our decentralized supply chain system.

```
var express = require('express');
var request = require('request');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
request('192.168.1.120:3000/clcSurface?a=2&b=3', function (error,
response, body) {
   if (!error && response.statusCode == 200) {
     console.log(body) // Print the output
   }
})
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

We used a module called request which is available via npm package manager. Request object enables us to make a request to other nodes by specifying their IP address with port on which server of the node we wish to call listens and a desired endpoint in our case `clcSurface` with parameters `a` and `b`. When the code is executed and if the other node is running it will return a response to our request and store it in the variable body which can be then printed to the console of caller node.

3. Beckhoff PLC platform and TwinCAT environment

3.1 Programmable logic controllers (PLC)

Programmable logic controllers (PLCs) [1] are industrial digital computers adapted for the control of manufacturing processes, such as assembly lines, robotic devices, or any activity that requires high reliability control, ease of programming and process fault diagnosis. They were built to replace hard-wire relays, timers, sequencers etc. and provide industry with a flexible and reliable controllers suitable for harsh environments. A PLC is a hard real-time system and output results must be produced in response to input signal within a limited time.

Typical components of the PLC among others include:

- **Central Processing Unit (CPU)**: This unit contains the brains of the PLC and is often referred to as a microprocessor. The basic instruction set is a high-level program, installed in Read-Only Memory (ROM). The programmed logic is usually stored in Electrically Erasable Permanent Read-Only Memory (EEPROM). The logic can be edited or changed if needed.

- **Memory**: System memory mostly implemented in flash technology is used by a PLC for a process control system. Aside from this operating system, it also contains a translated user program. Flash memory contents can be changed only in a case where the user program is being changed. Reprogramming a program memory is done through a serial cable in a program for application development. User memory is divided into blocks having special functions. Some parts of a memory are used for storing input and output status. Other parts of the memory are used to store variable contents for variables used in user programs. For example, timer value or counter value would be stored in this part of the memory.

- **Communication board**: Every brand of PLC has its own programming hardware, usually a computer based programmers. Computer-based programmers typically use a special communication board, installed in an industrial terminal or personal computer, with the appropriate software program installed. Computer-based programming allows offline programming, where the programmers develop their logic, store it on a disk, and then

download the program to the CPU. It also allows more than one programmer to develop different modules of the program. When connected to the CPU the programmer can test the system, and watch the logic operate when the system is running.

- **PLC controller inputs**: One of the most important ability of a PLC controller is to read signals from different types of sensors and input devices. The basic sensors include keys, keyboards, and functional switches, but there are also more specific automatic devices such as proximity sensors, marginal switches, photoelectric sensors, level sensors, inductive sensors for detecting metal objects, and so on. Input signals can be digital/logical (ON/OFF) or analog.

- **PLC controller outputs**: An industrial control system is incomplete if it is not connected with some output devices - actuators. Some of the most frequently used devices are motors, solenoids, relays, indicators, sound signalization, and so on. By starting a motor, or a relay, PLC can manage or control a simple system such as a system for sorting products all the way up to complex systems such as a service system for positioning the head of a robotic machine.

- **Power supply**: Electrical supply is used in bringing electrical energy to a CPU. Most PLC controllers work either at 24 V DC or 220 V AC. On bigger PLC controllers electrical supply comes as a separate module, while small and medium series already contain the supply module. Separate supplies need to be supplied to start PLC controller inputs and outputs. The internal logic and communication circuitry usually operates on 5 and 15 V DC power. Separate control transformers are often used to isolate inputs and CPU from output devices. The purpose is to isolate this sensitive circuitry from transient disturbances produced by any highly inductive output devices.

3.2 Staudinger warehouse model

Staudinger [2] compact high level storage warehouse model (Article no. 226001, Figure 3.1) will be used to enable storage functionalities of the distributed supply chain. The model simulates an automatically working high level storage system as used in many industrial branches. The model consists of a rack, divided up in a 3 x 3 storage places and a warehouse operating device, being portable in X direction. A cage, portable in Z directions, and including a telescopic palette carrier, that is portable iz Y direction, is attached to the warehouse operating device.

Table 3.1 presents warehouse model basic technical data. Modes of operation are enabled by a number of mechanical and reflection light sensors (switches) as well as a number of actuators (two direction motors and LED lights).

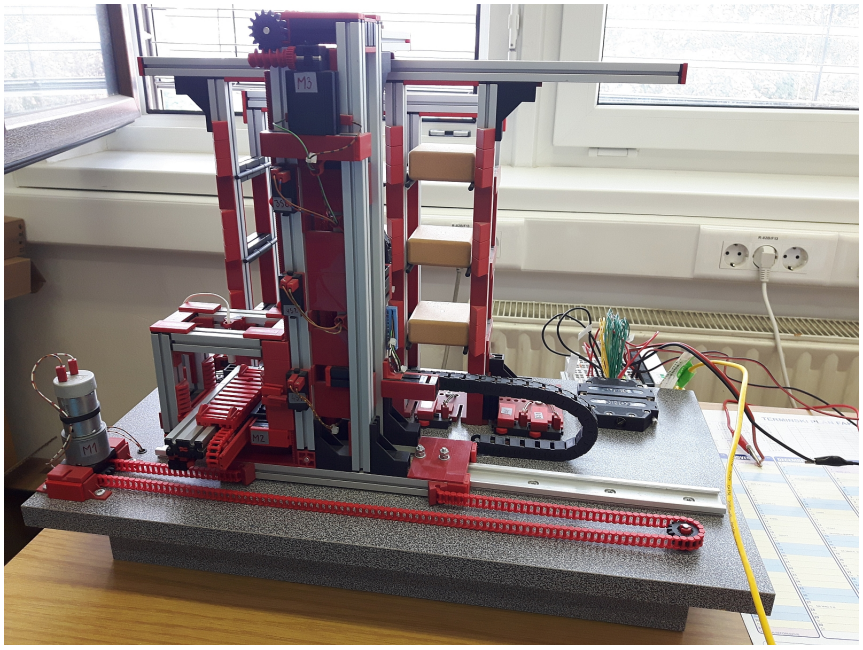

Figure 3.1: Staudinger Warehouse Model 226001

The model features two basic modes of operation:

- storing a package in the warehouse
- withdrawing a package from the warehouse

Technical data	**Value**
Power supply of sensors and actuators	24 V DC
Sensors	
Read switches	1
Mechanical switches	16
Actuators	
Motors with two directions	3
LED	2
Control system requirements	
Digital inputs (+ reading)	15
Digital outputs (+switching)	8

Table 3.1: Staudinger Warehouse Model 226001 Technical Data

Pin	Label	Function
Sensors		
1	3S1	X axis position 1 (X+)
2	3S2	X axis position 2
3	3S3	X axis position 3 (X-)
4	3S4	Y axis position Y+
5	3S5	Y axis position middle
6	3S6	Y axis position Y-
7	3S7	Z axis above position 1 (Z+)
8	3S8	Z axis below position 1
9	4S1	Z axis above position 2
10	4S2	Z axis below position 2
11	4S3	Z axis above position 3
12	4S4	Z axis below position 3 (Z-)
13	4S6	rack feeder occupied
14	3S7	hand key 1
15	3S8	hand key 2
16	spare	-
17	spare	-
18	X-	power supply 0 V
19	X-	power supply 0 V
Actuators		
20	5S1	X axis to X+
21	5S2	X axis to X-
22	7A1	Y axis to Y+
23	7A1	Y axis to Y-
24	7A2	Z axis to Z+
25	7A2	Z axis to Z-
26	5H1	LED light 1 (green)
27	5H2	LED light 2 (red)
28	spare	-
29	spare	-
30	spare	-
31	spare	-
32	spare	-
33	spare	-
34	spare	-
35	spare	-
36	2F1	power supply 24 V
37	2F2	power supply 24 V

Table 3.2: Staudinger Warehouse Model 226001 Sensors and Actuators

As presented in Table 3.1 15 sensors and 8 actuators are used to control the warehouse model. Two additional mechanical switches are installed at the both ends of the X axis to prevent the warehouse system from the fatal mistakes when using the conveyor or programming the control unit. These two switches (labelled 5S1 and 5S2) are working automatically and can not be directly accessed and manipulated.

Table 3.2 describes all sensors and actuators used in the warehouse model. This is also a list of connecting pins. Wiring from all the sensors and actuators in the model is merged into a 37 pin D-sub female connector. 37 D-sub male connector is needed to connect the warehouse to a PLC unit.

Figure 3.2 shows the warehouse layout with locations of all sensors and actuators as labelled in table 3.2.

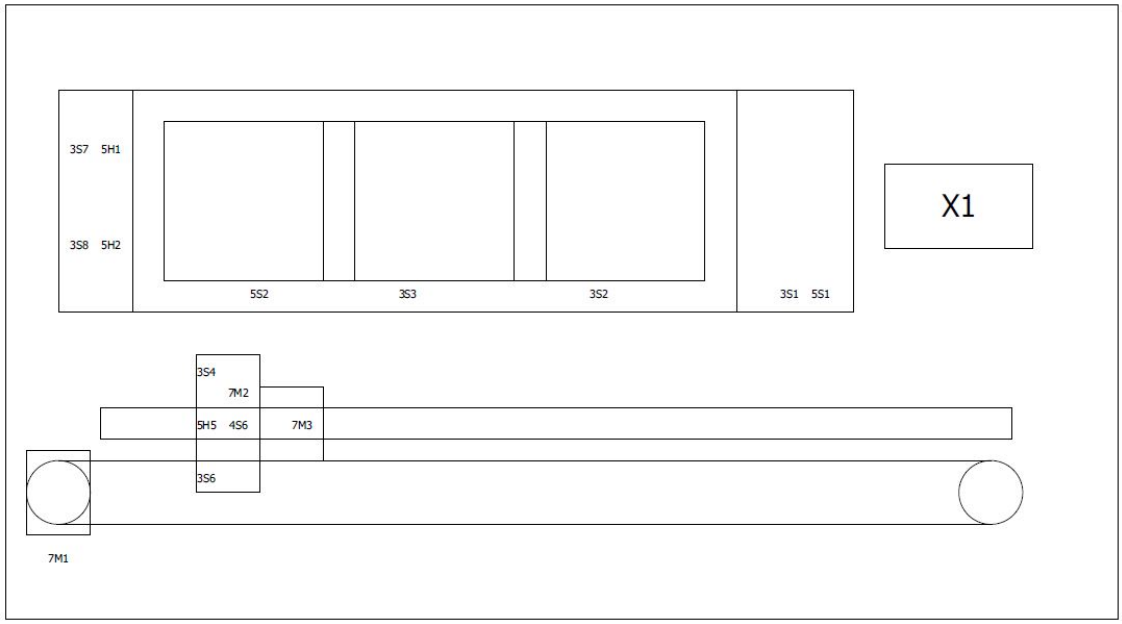

Figure 3.2: Staudinger Warehouse Model 226001 System Layout

3.3 Beckhoff automation and control systems

To enable control of the warehouse model, Beckhoff automation and control systems will be used. Beckhoff [3] implements open automation systems based on PC Control technology. Their product range covers industrial PCs, I/O and fieldbus components, drive (motion) technology and automation software. Beckhoff systems cover all fields of industries with products that can be used as a separate components or integrated into a complete and seamless control systems.

We will focus on using industrial PCs and input / output devices:

- **Industrial PCs**: PC components based on open standards and rugged construction of the device housings. Embedded PCs make modular IPC

technology available in miniature format for DIN rail mounting. Several types of PCs are available: embedded PCs, control cabinet PCs, panel PCs and control panels.

- **Input/Output**: Beckhoff supplies a wide range of fieldbus components for all common I/Os and fieldbus systems. In addition to conventional bus systems, Beckhoff offers its own solution, a complete EtherCAT I/O range for the high-speed Ethernet fieldbus based on EtherCAT terminals, the EtherCAT Box and EtherCAT Plug-in modules.

  - **Bus terminal**: open and fieldbus-neutral I/O system of electronic terminal blocks. The head of an electronic terminal block is the Bus Coupler with the interface to the fieldbus. Supported bus systems: **EtherCAT**, Lightbus, PROFIBUS, Interbus, CANopen, DeviceNet, ControlNet, **Modbus**, Fipio, CC-Link, SERCOS, RS232/RS485, Ethernet TCP/IP, EtherNet/IP, PROFINET, USB.

- **Automation**: Beckhoff offers comprehensive system solutions in different performance classes for all areas of automation. TwinCAT automation software integrates real-time control with PLC, NC and CNC functions in a single package. All Beckhoff controllers are programmed using TwinCAT software in accordance with the globally-recognised IEC 61131-3 programming standard. We will use TwinCAT 2 suite.

  - **TwinCAT 2**: automation software suite that forms the core of the control system. TwinCAT can turn almost any PC-based system into a real-time control with multiple PLC, NC, CNC and robotics runtime systems. More details about TwinCAT 2 can be found in Chapter 3.4.

3.3.1 Embedded PC CX 8090

Embedded PC CX 8090 (Figure 3.3) will be used for the Module 3 project assignment to manage and remotely control the warehouse. CX 8090 is a control system with a switched Ethernet port and supports protocols such as real-time Ethernet, ADS UDP/TCP, Modbus TCP client/server or open TCP/IP - UDP/IP communication. The CX 8090 is able to automatically recognise the type of I/O systems connected. The control system is programmed with TwinCAT via the fieldbus interface or the additional Ethernet interface. 24 V of DC power supply is needed to enable the functioning of the embedded PC and the connected I/O devices. More technical details are presented in table 3.3.

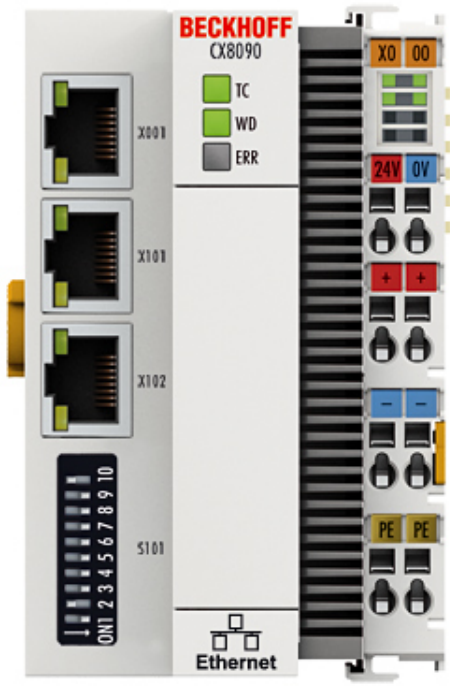

Figure 3.3: Beckhoff Embedded PC CX 8090

3.3.2 I/O devices

Two types of I/O devices will be used to enable the management and control of the warehouse. Potential distribution units will supply the warehouse with power, while digital input/output units will manage data exchange.

Potential distribution terminals

As already mentioned in Chapter 3.3.1 Beckhoff Embedded PCs run on 24 V of DC power. Besides powering the PC and connected I/O devices, it also provides means to power the connected PLC devices. Smart warehouse also runs on 24 V, so 0 V and 24 V potential distribution units will be used.

EL9186

The potential distribution terminal EL9186 provides eight terminal points with a potential of 24 V and enable the voltage to be picked up without further bus terminal blocks or wiring.

EL9187

The potential distribution terminal EL9187 provides eight terminal points with a potential of 0 V and enable the voltage to be picked up without further bus terminal blocks or wiring.

Technical data	**CX 8090s**
Processor	ARM9, 400 MHz
Internal main memory	64 MB RAM
Flash memory	512 MB RAM (expandable)
Protocols	real-time Ethernet, ADS TCP, Modbus TCP, TCP/IP, UDP/IP, EAP (EtherCAT Automation Protocol)
Programming	TwinCAT PLC
Programming languages	all languages defined in IEC 61131-3 standard
Web visualisation	yes
Online change	yes
Interfaces	1 x Ethernet 10/100 Mbit/s, 1 x USB device
Bus interface	2 x RJ45 (switched)
I/O connection	E-bus or K-bus, automatic recognition
Clock	internal battery-backed clock for time and date
Operating system	Microsoft Windows Embedded CE 6
Web based managementt	yes
Current supply	2A
Operating/storage temp.	0 to +55 °C / -25 to +85 °C
Relative humidity	95 %, no condensation
Protection class	IP 20

Table 3.3: Beckhoff CX 8090 Technical Data

Digital input / output terminals

Digital I/O terminals are used to acquire binary (true / false) control signals from device sensors or to send the control signals to the actuators. All I/O terminals have two power contacts (for 24 V and 0 V) that supply power to the terminal, while power supply is also looped through to the next terminal. E-bus or K-bus contacts and communication is used to transfer data. Terminals are automatically recognized and configured by the control PC and enable high interoperability. Up to 65535 units with robust housing and safe contacts can be assembled together.

EL1809

The 16-channel 24 V DC digital input terminal EL1809 acquires the binary control signals from the process level and transmits them, in an electrically isolated form, to the higher-level automation device. The EtherCAT Terminal contains 16 channels, whose signal states are displayed by LEDs.

EL2809

The 16-channel 24 V DC digital output terminal EL2809 connects the binary control signals from the automation device on to the actuators at the process level

with electrical isolation. The EL2809 is protected against polarity reversal and processes load currents with outputs protected against overload and short-circuit. The EtherCAT Terminal contains 16 channels, whose signal states are displayed by LEDs.

3.4 TwinCAT

Beckhoff developed a specialized software to manage their control and automation systems. TwinCAT, The Windows Control and Automation, is a software system that turns any compatible PC into a real-time controller with a multi-PLC system, programming environment and operation station. TwinCAT replaces conventional PLC controllers and operating devices with an open and compatible PC hardware and embedded software PLC based on IEC 61131-3 standard for PLC programming. TwinCAT includes programming and run-time systems and offers connection to all common fieldbuses. PC interfaces are supported as is data communication with user interfaces and other programs by means of open Microsoft standards (OPC, OCX, DLL, etc.).

The latest version of TwinCAT is TwinCAT 3, but as LDSE Summer School Module 3 assignment will be implemented in TwinCAT 2, we hereafter shortly present basic features of TwinCAT version 2 [4].

TwinCAT 2 software suite includes various components that enable CNC, NC PTP and NC Interpolation functionalities, and drivers for Windows I/O and control panels. The two most important components, used in any PLC application, are:

- **TwinCAT System Manager**: the configuration centre for the system, relationships are defined among the number of PLC systems, PLC system programs, configuration of axis control and connected I/O channels.
- **TwinCAT PLC Control**: PLC programming environment, code is programmed in accordance with IEC 61131-3 standards independently of the manufacturer; PLC Control supports all languages of the IEC standard and enables online connections with PLC runtime systems around the world with TCP/IP or via fieldbuses.

3.4.1 TwinCAT Usage

To easier understand how TwinCAT is used to develop and run control program for the warehouse, we hereafter briefly describe operations needed to set up a working PLC program with TwinCAT.

- PLC programs are developed on a remote working station (PC or laptop).
- Both Embedded PC (CX 8090) and remote working station must be running the TwinCAT software.

- **TwinCAT System Manager** is used first to set up a complete configuration.
- TwinCAT System Manager - System Configuration connects the remote working station to the embedded PC that controls the warehouse.
- TwinCAT System Manager - I/O Configuration detects, recognises and configures I/O devices connected in the embedded PC assembly.
- TwinCAT System Manager - PLC Configuration adds and configures PLC program that is run to control the warehouse.
- **TwinCAT PLC Control** is used next to develop the control program, including the input and output variables mapped to the physical connectors on the I/O devices and programming logic that enables all modes of operation.
- After the control program is developed, checked and compiled, TwinCAT PLC Control is used to download the program on the embedded PC.
- Using TwinCAT PLC Control the control program is remotely run, stopped, and, if needed, debugged.
- Modbus TCP feature of the CX 8090 is used to connect to the embedded PC via Ethernet, read state of the sensors and set the actuators.

A short tutorial on how to use TwinCAT System Manager and TwinCAT PLC Control follows in the next two chapters.

3.4.2 System Manager

TwinCAT System Manager [5] is a program for general system configuration. It enables connection from remote working station to the PC running the PLC program, detection and configuration of connected I/O devices and setting up a PLC program that controls the operations.

TwinCAT System Manager usage includes the following operations:

- Open new System Manager file.
- Connect to the control PC.
- Add a PLC project.
- Add I/O devices and generate mappings from PLC program variables and physical connectors of I/O devices.
- Check configuration.
- Activate configuration.
- Set System Manager to run mode.

Open new System Manager file

In System Manager main program menu select *File* → *New* (Figure 3.4). This will open new *.tsm* (TwinCAT System Manager) file. Save this file for future reference and give it an appropriate name.

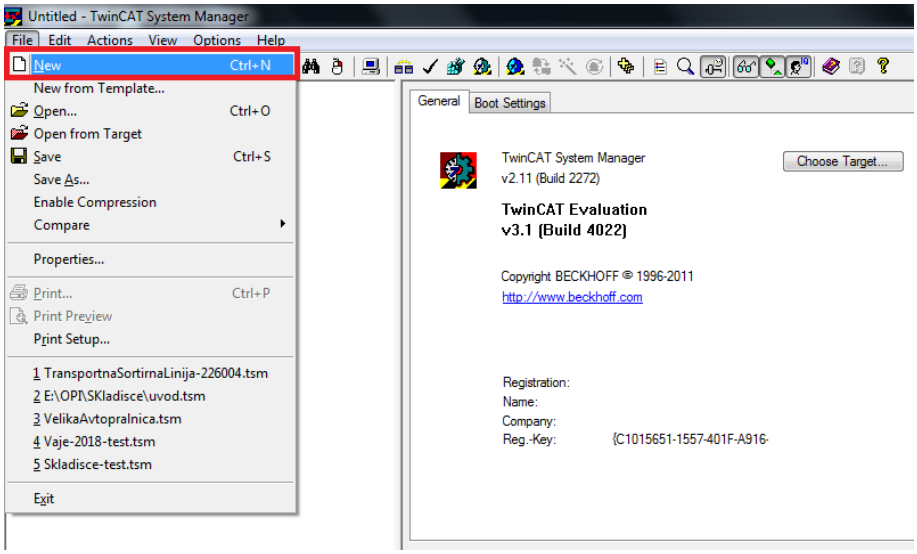

Figure 3.4: TwinCAT System Manager - open new file

Connect to the control PC

To successfully fulfil this step your remote working station needs to be physically connected to the control PC (CX 8090) with the Ethernet cable. From the program window on the left select *SYSTEM - Configuration*. From the program window on the right select the *Choose Target* button (Figure 3.5).

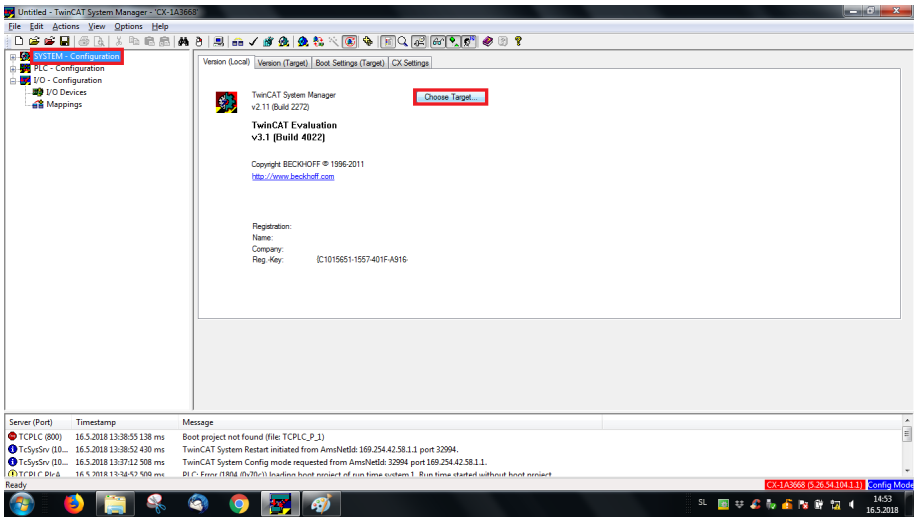

Figure 3.5: TwinCAT System Manager - System Configuration - Choose Target

A *Choose Target System* pop-up window appears. Select the *Search (Ethernet)* button.

An *Add Route Dialog* pop-up window appears. Select *IP Address* option in the *Address Info* section. Now press the *Broadcast Search* button.

The connected embedded PC will appear on the search results list with a host name *CX-aaaaaa*, where *'aaaaaa'* is a combination of numbers and upper case letters.

Select the detected embedded PC and press the *Add Route* button. A pop-up window appears to enter user credentials to log on the target system. Enter:

- user: **Administrator**
- password: **1**

and press OK. This will create a route from the working station to the control PC and connect both devices. A letter **X** appears next to the host name of the embedded PC in the *Connected* section of the search results list.

Press close to return to the previous window. The connected CX 8090 device appears on the list of possible target systems. Select it and press OK. Remote working station should now be connected to the target system.

Add a PLC project

From the program window on the left select *PLC - Configuration* and right click on it. Select *Append PLC Project* (Figure 3.6). An *Insert IEC1131 Project* window appears. Find and select a *TwinCAT Project Info* file (*.tpy*). It must be previously created to add it in this step. TwinCAT Project Info file is created in TwinCAT PLC Control program after creating and building a PLC project. Refer to Chapter 3.4.3 for more information on how to create *.tpy* file.

Check to see if the correct *target system* was chosen (CX embedded PCs run on ARM architecture) and what are the *run-time number* and *port*.

The selected PLC project appears in the left program windows in the *PLC - Configuration* section. Expand it to check various options. Under the *Standard* option you will see the *Inputs* and *Outputs* sections.

Input and output variables that you define in the control program to work with the controlled device will appear in this sections and need to be linked to the physical connections of the appropriate I/O devices. From the list of variables select it, one by one, right click on it and select *Change Link*. A list of available addresses on the I/O devices will appear, select the correct one.

Add I/O devices

This part of system configuration works in *Config* mode of TwinCAT. From the toolbar select the *Set/Reset to Config* mode icon or press *Shift + F4* (Figure 3.7) .

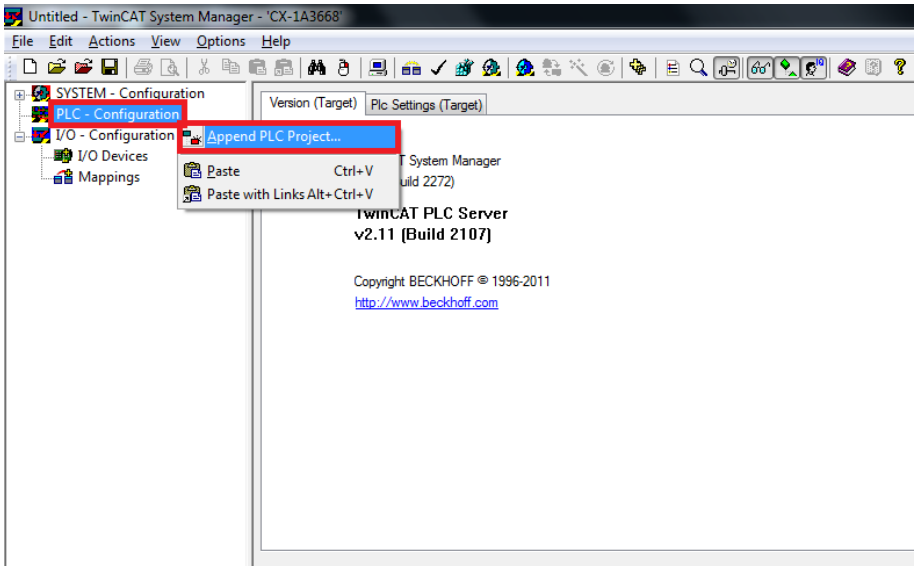

Figure 3.6: TwinCAT System Manager - PLC Configuration - Append PLC Project

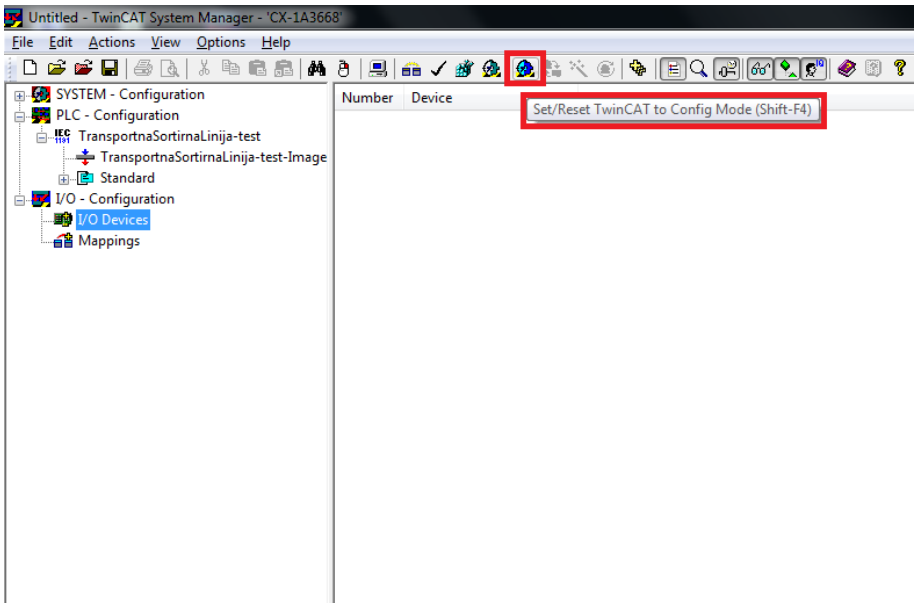

Figure 3.7: TwinCAT System Manager - I/O Configuration - Set TwinCAT to Config mode

From the program window on the left select *I/O - Configuration*. Select *I/O Devices* and right click on it. Select the *Scan Devices* option and then click OK when a pop-up window appears informing you that some devices may not be found automatically. This process first finds all primary devices (which include EtherCAT devices and internal PC RAM) and then goes on to find I/O boxes (connected terminals).

A list of discovered devices appear in the program window on the left under *I/O devices*. Expand the *Device 1 (EtherCAT)* to see the terminals found on this

device. It first finds *Term 1 (EK1200)* terminal which is an EtherCAT bus and if you expand it even further you will discover three more terminals attached to the EtherCAT bus. *Term 2* is digital input terminal (EL1809), *Term 3* is a digital output terminal (EL2809) and *Term 4* is a bus end terminal (EL9011).

TwinCAT I/O configuration offers an option of directly reading and changing input and output signals from the connected terminals. After all the devices and boxes have been discovered System Manager asks you if you want to enable *Free Run* option (still in Config mode). With this option enabled go to *Term 4* (output terminal), expand it and select *Channel 8* (led light). Expand it again and select *Output*. In program window on the right select the *Online* tab. Current value of the output signal is shown in the Value field and to change it use the *Write* button. One of the LED lights on the warehouse model will turn on.

After all the I/O terminals have been discovered and configured (this is done automatically), you need to generate mappings between PLC project variables and I/O devices channels. Select *Generate Mappings* from the *I/O - Configuration* option in the left program window of the System Manager.

Check and activate configuration

After all the configuration has been done you need to check and activate it. Use icons on the toolbar (Figures 3.8 and 3.9). If everything is OK, the System Manager asks you if you wish to set the TwinCAT in run mode. A successful configuration check and setting the TwinCAT in run mode is confirmed with a green colour of the status bar in the right bottom corner of the System Manager program window.

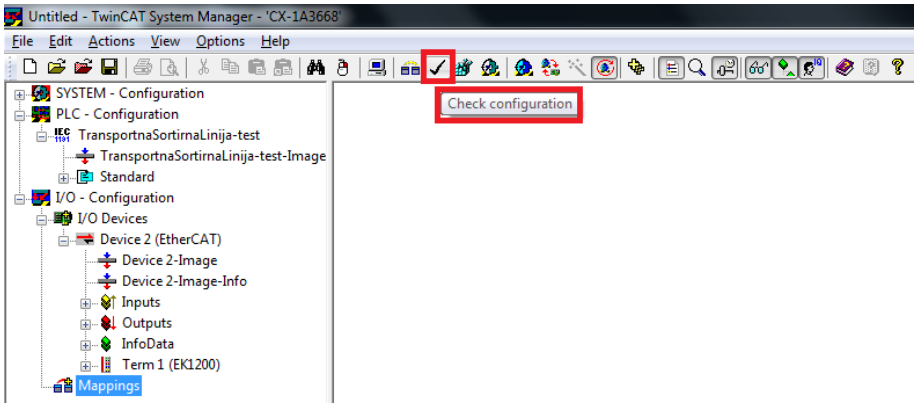

Figure 3.8: TwinCAT System Manager - I/O Configuration - Check Configuration

3.4.3 PLC Control

TwinCAT PLC Control [6] is a PLC project development environment, based on IEC 61131-3 standard for PLC programming. It supports all programming languages of the IEC standard, we will focus on a high-level textual language Structured Text (more details in Chapter 3.5).

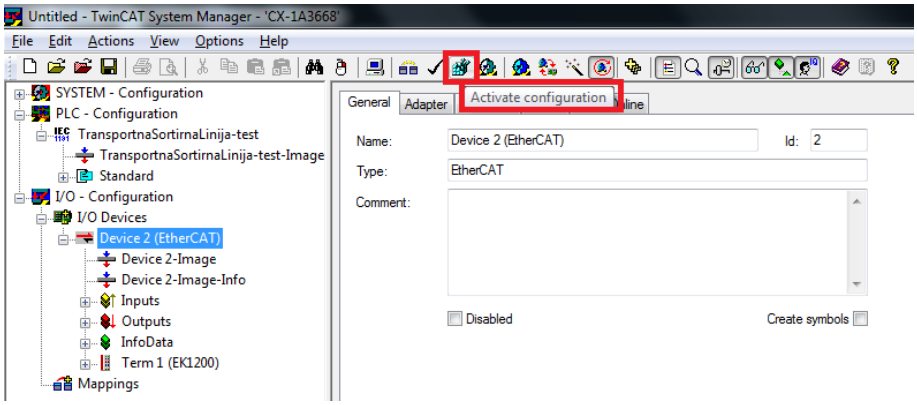

Figure 3.9: TwinCAT System Manager - I/O Configuration - Activate Configuration

TwinCAT PLC Control program window is divided into several parts and sub-windows (Figure 3.10). Menu bar allows access to all the functionalities of the software while most important functions of the program are also accessible from a toolbar below the menu bar using small icons.

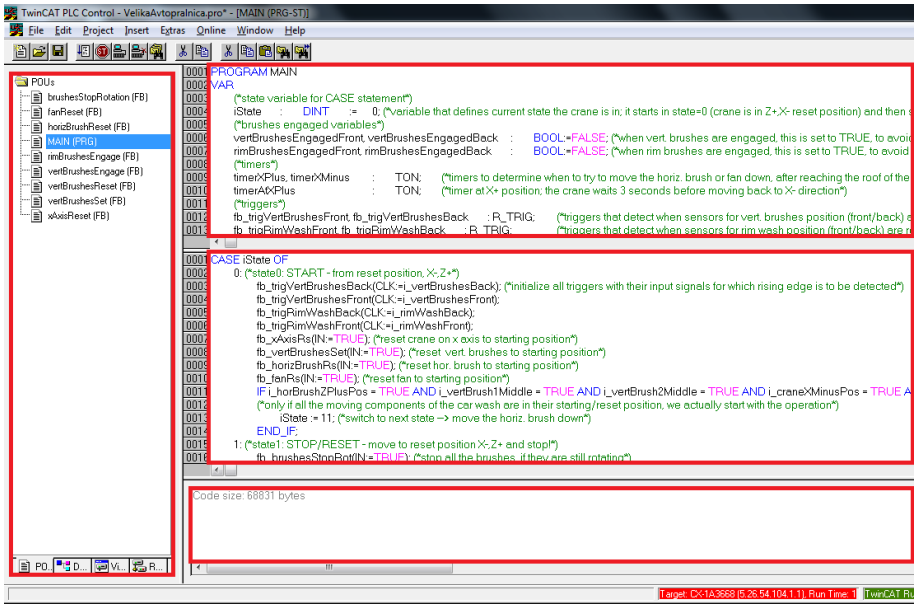

Figure 3.10: TwinCAT PLC Control Overview

Main PLC Control program sub-windows are:

- On the left: POUs / Data Types / Visualizations / Resources window - all program organization units (POUs) are listed here. The main POU is MAIN (Program) and other POUs such as functions and function blocks can also be defined here. This window is also used to access additional resources, such as global variables, libraries etc.
- In the top centre position: start of the POU (program, function or function block) with a type of the POU, name of the POU and declaration and

initialization of the local variables. Functions' input variables are also defined here, as are function blocks' input and output variables.

- In the middle centre position: main programming window, all the program code goes here (variables' values assignment, programming logic etc.)

- In the bottom centre position: log window with compiler logger information, such as loading a library, compiling, warnings and errors in the code etc.

Program Organization Units (POUs)

Program organization units are core building parts of the PLC application. Three types of POUs are defined:

- Program

- Function

- Function block

For simple PLC applications all the code can go into a program (like MAIN), while functions and function block are useful when the code gets larger and some functionalities of the application repeat. Functions and function blocks are used to make the code cleaner and easier to program.

Basic difference between a function and a function block is that given the same parameter values a function always returns the same value, while a function block output depends on the state of internal variables from previous invocation of the function block. FB output is not necessarily the same every time it is invoked. Function blocks must first be created as a "class" objects with input, internal and output variables, then instantiated and receive an instance identifier. Function block have output variables, that can be accessed at any time. Functions, on the other hand, do not have outputs, but rather a return value as the function identifier is actually a "container" for the result value, so it must be assigned a data type.

Many predefined functions and function blocks are available from libraries and custom functions and function block can also be defined, as applicable in a specific PLC application. Here are some examples of the standard functions and function blocks:

- **functions**: *ADD*, *SQRT*, *SIN*, *COS*, *GT*, *MIN*, *MAX*, *AND*, *OR* etc.

- **function blocks**: *TON*, *TOF*, *TP* (timers), *CTD*, *CTU*, *CTUD* (counters), *F_TRIG*, *R_TRIG* (triggers) etc.

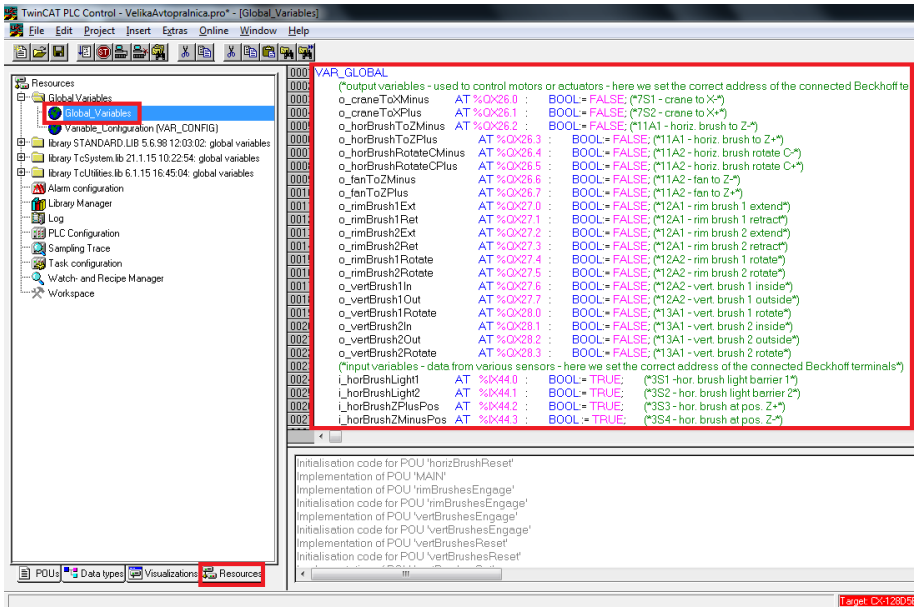

Figure 3.11: TwinCAT PLC Control - Global variables

Variables

Variables are an essential part of any programming language. PLC programming languages define several types of variables:

- **global** : accessible from any part of the PLC application (see Figure 3.11)
- **local**: accessible inside a POU (program, function, function block) (see Figure 3.12)
- **I/O**: a local or a global variable mapped to input or output signal from a controlled device
- **static** and **temporary**: static variables' values are stored to the internal DB and can be used later when a function block is invoked again; temporary variables' values are not stored and are lost after a function or function block is exited
- **external**: external variables are global variables that are imported into a function block

Some variable attributes can also be defined:

- **AT**: set variable at specific location (for example, input or output location, internal memory location etc.
- **RETAIN**: retain variables retain their value after an uncontrolled stop (after a reset or if the PLC program is unexpectedly downloaded to the control PC again); when the program restarts, the system continues to operate with the stored values

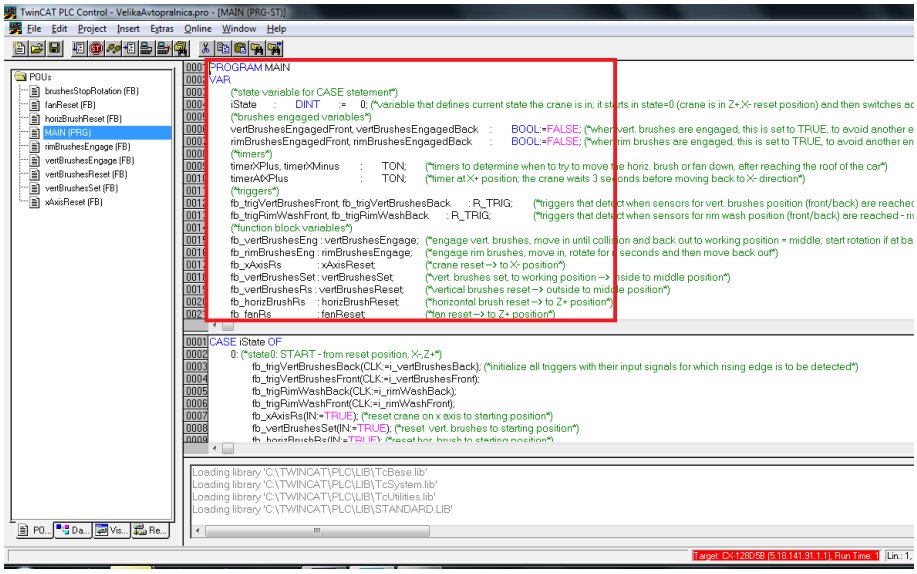

Figure 3.12: TwinCAT PLC Control - Local variables

- **PERSISTENT**: similar as retain, but persistent variables are only retained when the TwinCAT shuts down
- **CONSTANT**: variables that are constants, i.e. their values can not be changed

Libraries

Many useful functions and function blocks have already been written and it is a good programming practice to reuse them wherever possible. Libraries can be include in a PLC application via a *Library Manager* (see Figure 3.13). *Standard.lib* library containing declarations of timers, counters, triggers and string function blocks is automatically included and other libraries such as *TcSystem.lib* and *TcBase.lib* can be added (right click in the area where libraries are listed and select *Additional Library*).

Main programming window

The main part of the code is written here. Depending on the PLC application that is programmed, appropriate language constructs, data types, decisions, loops etc. are used. More details about writing PLC application in a Structured Text (ST) programming language are presented in Chapter 3.5, here we present just an excerpt from a real-life application for you to get an insight how a PLC application looks like (Figure 3.14).

A special note must be made about programming a PLC application. Usually, a program is written in such a way that it is started and run only once and when all the program steps (code statements) are executed the program stops and exists. A PLC

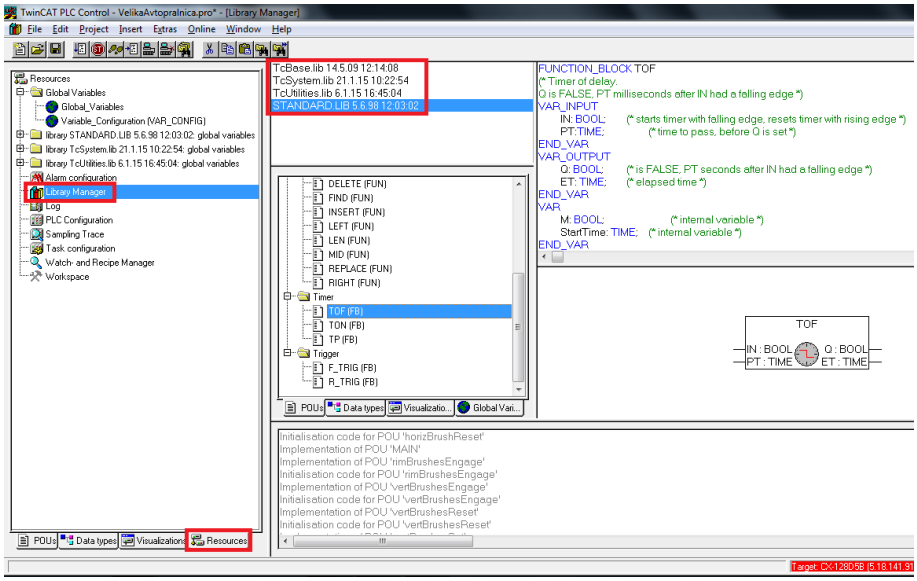

Figure 3.13: TwinCAT PLC Control - Library Manager

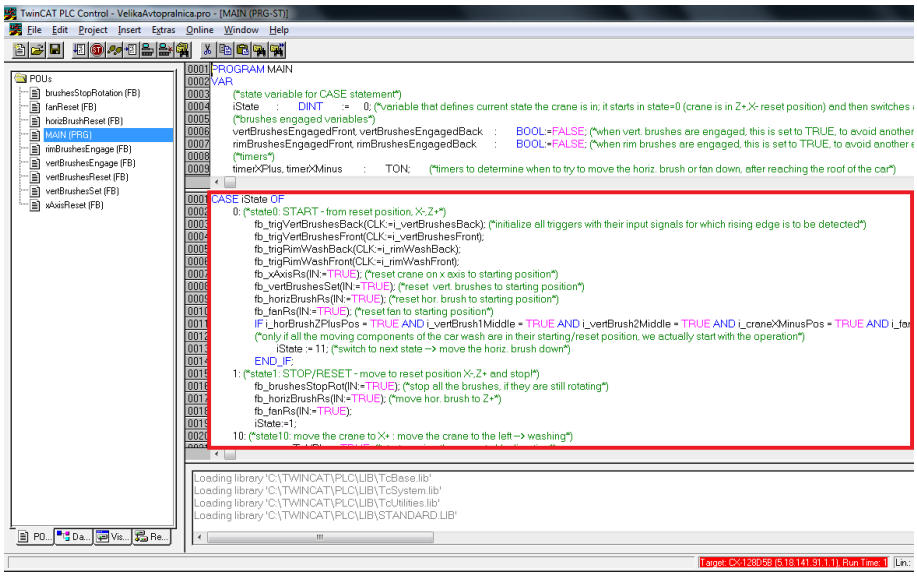

Figure 3.14: TwinCAT PLC Control - Main Programming Window

application must be programmed differently. It is run periodically and it basically never stops. It constantly checks the state of the device sensors and, depending on the programming logic, reacts to a change by activating some of the actuators. TwinCAT does that by going through a code and executing program statements over and over again until the application is stopped manually. Program repetition frequency is defined by PLC Cycle (Base) time in TwinCAT System Manager, inside *SYSTEM - Configuration* (*Real-Time Settings*).

Log window

Logging window at the bottom of the PLC Control application displays notifications from a compiler, including various tasks it performs during a compilation, external libraries it includes in the project and if, when checking the code, it encounters some problems. The compiler displays warnings and errors that need to be corrected.

TwinCAT PLC Control Usage

Using TwinCAT PLC Control involves several steps, from programming a PLC code, building the code and creating a translated version of the code specific to a control PC, logging on a control PC and running the code. We will go through some of the details of each step:

- **Create new PLC project**: Select *File* → *New*. A *Choose Target System Type* dialog window appears. Beckhoff CX 8090 embedded PC which is based on ARM processor architecture will be used a control PC, so select *CX (ARM)*. *New POU* dialog window now appears where we create a new main POU (*Program*) and select *ST* as a language of the POU. By clicking OK a *TwinCAT PLC Project* file (*.pro*) is created. Save it with an appropriate file name. Next, run-time system must be selected that will run the PLC application. Select *Online* → *Choose Run-Time System*. A *Choose Run-Time System* dialog window appears where you select the control PC and an available run-time system within the PC. Note: the remote working station must be physically connected to the control PC at this point and a proper connection must also be made in the TwinCAT System Manager Configuration (see Chapter 3.4.2).

- **Programming the PLC application**: Write the application in the appropriate program windows - use the top centre window for declaration of variables and the middle window for the main part of the code. Use appropriate coding syntax, data types, programming constructs etc. (for more information on how to properly program in Structured Text programming language see Chapter 3.5).

- **Building the code**: Select *Project* → *Build*. This will check the code for possible mistakes and show where they are. If (or when) the code is syntactically OK, the building process results in a translated code that suits the architecture of the target system. Building process also generates a *TwinCAT Project Info* file (*.tpy*) that is added into a PLC Configuration of the TwinCAT System Manager. Adding the *.tpy* file to a System Configuration automatically recognizes I/O variables in the program and lists them in the configuration. These need to be linked to a physical inputs and outputs of the PLC hardware assembly. Once a program is built and added to the System Configuration for the first time, all further changes in the program are immediately visible in

System Configuration. No actions in System Configuration are needed if the program changes, except if the I/O variables change.

- **Connecting to the control PC**: Select *Online* → *Login*. This connects the remote working station to the control PC. If there is no PLC application on the control PC, the pop-up window appears asking you if you want to download a program from remote station to the control PC. If there is an application on the control PC, the pop-up window appears informing you that *Online Changes* will be made to the PLC application. After a successful login the TwinCAT PLC Control window in the middle of the screen changes and shows program variables' current values. If you want to log out the control PC select *Online* → *Logout*. Note: always stop the application and reset the variables' values before logging out to avoid unexpected movements of the controlled device and a possible damage.

- **Running the PLC application**: When logged in select *Online* → *Run* or press *F5* to start the application. To stop the application select *Online* → *Stop* or press *Shift + F8*. If you want to analyse the application step by step first toggle some breakpoint(s) by selecting *Online* → *Toggle Breakpoint* and then run the application. The application will stop at breakpoint(s) and allow you to go through the code step by step. To step over a line of code select *Online* → *Step over* or press *F10* and if you want to move inside a function or function block call select *Online* → *Step in* or press *F8*.

3.5 Structured Text (ST)

Structured Text will be used to program control application for the warehouse. Structured Text is a high level textual programming language that is block structured and syntactically resembles Pascal, on which it is based. ST supports complex statements and nested instructions, such as decisions (conditional execution), iteration loops and functions. Structured Text is of the standardized languages for PLC programming and is a part of IEC 61131-3 standard [7].

IEC 61131-3 is the third part (of 10) of the open international standard IEC 61131 for programmable logic controllers, and was first published in December 1993 by the IEC. The current (third) edition was published in February 2013.

Part 3 of IEC 61131 deals with basic software architecture and programming languages of the control program within PLC. It defines two graphical and two textual programming language standards:

- Ladder diagram (LD), graphical
- Function block diagram (FBD), graphical
- Structured text (ST), textual

- Instruction list (IL), textual
- Sequential function chart (SFC) - has elements to organize programs for sequential and parallel control processing.

For more information on PLC programming with Structured Text language check [8].

3.6 Modbus TCP

Modbus [9] is a serial communications protocol designed for use with programmable logic controllers. It was developed in 1979 and is now a de facto standard communication protocol used in many industrial devices. It was specifically designed with industrial applications in mind, it is openly published and easy to deploy and maintain. It is based on master/slave architecture where there is a master (supervisory) computer and a slave remote terminal unit. A good advantage of this protocol is also that it moves raw bits or words without placing many restrictions on vendors.

Many variants of the Modbus protocol exist. The most common implementation is **Modbus RTU** (remote terminal unit). This version is used in serial communication and uses a compact binary representation of the data for protocol communication. **Modbus TCP** [10] (or Modbus TCP/IP) variant is used for communication over TCP/IP network and connects over port 502. Modbus TCP will be used in the project assignment to connect to and control the warehouse.

Modbus defines 4 basic object types that are provided by a Modbus slave device to a Modbus master device (see Table 3.4). A coil is a single-bit physical output which can be both read and written to. A discrete input is a single-bit physical input and it can only be read. An input register is an object type that allows to read 16 input signals simultaneously, while a holding register allows to write to 16 output signals simultaneously.

Object type	**Access**	**Size**	**Use**
Coil	read and write	1 bit	on / off
Discrete input	read only	1 bit	on / off
Input register	read only	16 bits	measurements and statues
Holding register	read and write	16 bits	configuration values

Table 3.4: Modbus Object Types

Various operations defined in Modbus are categorized with a function code. These operations include data access, diagnostics and some other operations. Here, we focus on data access operations. Table 3.5 presents the most basic and commonly

used data access operations - reading and writing single or multiple physical inputs and outputs.

Function code	Function name	Access type
2	Read Discrete Inputs	bit access
1	Read Coils	bit access
5	Write Single Coil	bit access
15	Write Multiple Coils	bit access
4	Read Input Registers	16-bit access
3	Read Multiple Holding Registers	16-bit access
6	Write Single Holding Register	16-bit access
16	Write Multiple Holding Registers	16-bit access

Table 3.5: Modbus Basic Functions

3.6.1 Modbus TCP in Beckhoff systems

Beckhoff offers several solutions for using Modbus protocol in their systems. Aa specialized Embedded PC for Ethernet - CX 8090 (see Chapter 3.3.1) will be used in Module 3 assignments. CX 8090 natively supports Modbus TCP and no extra installation or configuration is needed to make Modbus TCP protocol work on CX 8090. CX 8090 Modbus TCP protocol support provides both server and client functionalities. Modbus registers and I/O's are automatically mapped to TwinCAT PLC areas (variables) and a supplied PLC library allows to communicate with other Modbus devices to request data.

The default mapping between Modbus and TwinCAT PLC areas is described in Table 3.6. Let's look at some examples to easier grasp the principle of Modbus addressing.

Let's assume we have a CX 8090 terminal assembly with one digital input terminal EL 1809 and one digital output terminal EL 2809 3.3.2. Both terminals have 16 physical connections. When configuring both I/O devices in the TwinCAT System Manager the TwinCAT assigns specific addresses to all the physical connections. As both terminals are digital, assigned addresses are in a binary form. For example, digital input terminal physical connections are assigned addresses from *40.0* to *40.7* and from *41.0* to *41.7*. Digital output terminal physical connections are assigned addresses from *26.0* to *26.7* and from *27.0* to *27.7*. The decimal number in the first part of the address represents number of bytes, while the last digit in the address represents bits in the last byte.

To **read values of sensors** (digital input terminal) from Modbus TCP client we use the function *Read Discrete Inputs* with addresses from *320* to *335* (if used in a decimal form) or from *140* to *14E* (if used in a hexadecimal form). Where do

Modbus area	**Modbus addresses**	**TwinCAT area**
Discrete inputs	0x0000 - 0x7FFF	0xF021 - process image of the physical inputs (bit access)
Coils	0x0000 - 0x7FFF	0xF031 - process image of the physical outputs (bit access)
Input registers	0x0000 - 0x7FFF	0xF020 - process image of the physical inputs)
Holding registers	0x0000 - 0x2FFF	0xF030 - process image of the physical outputs
	0x3000 - 0x5FFF	0x4020 - PLC memory area
	0x6000 - 0x7FFF	0x4040 - PLC data area

Table 3.6: Modbus to TwinCAT PLC Mapping

this numbers come from? First, as presented in table 3.6 discrete inputs start at Modbus address *0 (0x0000)* so there is no offset. The address *40.0* is in bytes form, but we need it in bit form. So: $40 \cdot 8 + 0 = 320$ bits. Similarly *41.7* in bytes form is $41 \cdot 8 + 7 = 335$ bits. *140* and *14E* are hexadecimal representations of *320* and *335*.

To **control the actuators** (start or stop them) we use the function *Write Single Coil* with addresses from *208* to *223* (decimal form) or *D0* to *DF* (hexadecimal form). Again, coils start at Modbus address 0 so there is no offset. The address *26.0* is in bytes form, but we need it in bit form. So $26 \cdot 8 + 0 = 208$ bits. Similarly *27.7* in bytes form is $27 \cdot 8 + 7 = 223$ bits. *D0* and *DF* are hexadecimal representations of *208* and *223*.

To **read or write to a variable in the PLC memory area** (if the M attribute is used in the variable declaration, for example: `var AT \%MX0.0 : BOOL := TRUE`) we use the functions *Read Holding Register* or *Write Holding Registers*. Addresses from *12288* (decimal form) or from *3000* (hexadecimal form) on must be used.

To **read or write to a variable in PLC data area** (if not extra attribute is used in the variable declaration) we use the functions *Read Holding Register* or *Write Holding Registers*. Addresses from *24576* (decimal form) or from *6000* (hexadecimal form) on must be used. Two registers (each is 16 bits long) are reserved for each variable.

A 2018 Summer School Project Assignments

Summer school is set in a way that participants have the main role in designing the nodes and communications between them. One of the most representational graphic tool for designing communication between nodes is a Message Sequence Chart (MSC). MSC in Figure A.1 shows all four types of nodes and a customer that is there only as a free agent as it does not represent a node.

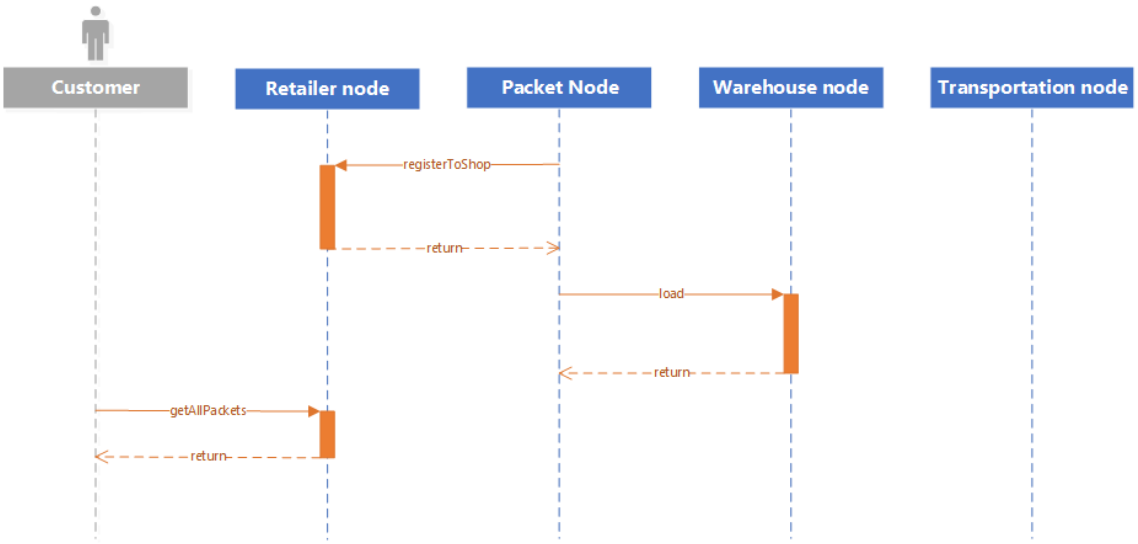

Figure A.1: MCS of distributed supply chain system

MSC shows an example process of adding a new package into the system. In the first step, the package node requests to be registered on retailer node by sending a request message `registerToShop`. This means that package node sends a message to the retailer node Web API endpoint that enables a package to be registered and can appear in the retailers online shop. In the second step, a package node requests a warehouse node to store the package by sending a message `load`. When the warehouse node loads the package it returns, for example, a message of the package location in case of success, or rejects the package if the warehouse is full.

A.1 Module 1: Smart Package Node

Demands for building a smart package node are:

1. Define package node properties:

   - id: Identification string of the package. It has to be unique for each package.
   - owner: Owner of the package is user that buys the package.
   - content
     Describes content of the package.
   - type: Type defines special treatment of the package based on its content like food, liquid, glass etc.
   - location: This parameter shows current location of the package in the form of the IP address of the warehouse node.
   - size: Size of the package in the form: Length x Width x Height (in mm).
   - state: State defines the current state of the package. States of the package are:
     initial: the package server has started but the package is not yet created
     created: the package properties has been set
     registered: the package has been registered with the retailer node
     purchased: the package has been purchased by some user
     transport_ordered: transport of the package has been ordered
     transport_pickup: transport is on its way to pick up the package
     warehouse_unloading: transport has arrived at the warehouse location and is waiting for the package to be unloaded
     warehouse_unloaded: package is loaded to the transportation
     transport_delivering: package is on its way to the customer
     transport_delivered: package has been delivered to the customer

2. Build WEB API endpoints

   - `createPackage`
     Create package with all the package parameters. Example: `http://192.168.1.100:3000/createPackage?content=banane&owner=Janez&type=glass`

   - `registerToMarketPlace`
     Call retailer node and register to its online shop. Example: `http://192.168.1.100:3000/registerToMarketPlaces?reatailer_ip=192.168.1.101`
     Because the package node will communicate with the retailer node we have to provide retailers node's IP address.

- `load`
  initial load of package to the warehouse. Example `http://192.168.1.100:3000/load?warehouse_ip=192.168.1.102`
  Because the package node has to communicate with the warehouse node we have to provide warehouse's IP address.

3. Deploy 3 packages (optional)
4. Include also economic aspects into the system. (optional)

A.2 Module 2: Retailer Node

Demands for building a retailer node are:

1. Define retailer node properties
   - packages: Array of packages that can be filled with packages data that are then displayed in online store web page.
2. Build WEB API endpoints
   - `initialize`
     Initialize the retailer shop on startup of the system. Example: `http://192.168.1.101:3000/initialize`
   - `getPackages`
     Return all packages that are registered in the shop.
   - `removePackage`
     Remove package from the retailers memory of items that are offered in the online store. Example `http://192.168.1.101:3000/removePackage?id=gfwqgebgwtegvwe`
     Remove package with id "gfwqgebgwtegvwe" from the retailer node memory.
   - `buy`
     Buy a package with a certain id. When customer visits online store and buys one product this web API is called. Example: `http://192.168.1.101:3000/buy?id=gfwqgebgwtegvwe`
   - `register`
     Register package. `http://192.168.1.101:3000/buy?id=gfwqgebgwtegvwe&content=banane&owner=Janez&type=glass&location=192.168.1.102`
     In order to register package to the retailer node we have to append all data of the package.
3. Build shop web page (optional).
4. Include also economic aspect into the system. The retailer charge the package a fee for being listed or additional services such as advertisement. (optional)

A.3 Module 3: Warehouse Node

Demands for building a warehouse node are:

1. Define warehouse node properties:
   - location: location of the warehouse (longitude and latitude coordinates)
   - occupancy: storage capacity of the warehouse
   - orders: array of collected orders for warehouse (load / unload)
   - occupation information
   - state:
     loading: loading a package into the warehouse
     unloading: unloading a package from the warehouse
     idle: no operation

2. Build WEB API endpoints
   (a) `loadPackage`
   Load a package into the warehouse. Smart Package node calls this API when a new package (product) is registered in the system and needs to be stored in a warehouse. Example: `http://192.168.1.103:3000/load?id=gfwqgebgwtegvwe`
   (b) `unloadPackage`
   Unload a package from the warehouse. Transport node calls this API when it receives an order from the smart package node to deliver the package to a customer. Example: `http://192.168.1.103:3000/load?id=gfwqgebgwtegvwe`

3. Include also economic aspects into the system. The warehouse node charges the package a fee for storage and for loading/unloading services. (optional)

In this module you will first create a PLC application for warehouse control and then a warehouse node with logic for communication with the warehouse and APIs for integration into the a distributed supply chain system.

Assignment 1: PLC Project Setup

Use TwinCAT System Manager to set up a PLC project for warehouse management. For more information on how to use TwinCAT System Manager, see Chapter 3.4.2. Connect your remote working station to the Embedded PC CX 8090 which is connected to the warehouse and will run the PLC project code.

First, create and save a System Manager file (.tsm file). This file will contain complete PLC project configuration. PLC project configuration should include:

- System Configuration:
  - Connect your remote working station to the embedded PC CX 8090 that will run the PLC project code.
  - *Choose Target*
  - *Search (Ethernet)*
  - *Broadcast Search* (select *IP Address* option in the *Address Info* section)
  - *Add Route* (user name: Administrator, password: 1)
- PLC Configuration:
  - In TwinCAT PLC Control create and save a new PLC Project (.pro file) - Target System Type is *CX (ARM)*, programming language is *Structured Text*. For testing purposes declare a variable and assign it some value.
  - Build the project and create .tpy file
  - In System Manager add .tpy file in the PLC Configuration.
- I/O Configuration
  - Run System Manager in **CONFIG** mode to find devices and terminals attached to the control PC.
  - *I/O devices* → *Scan Devices*
  - *Scan boxes*
  - *Free run*
  - Use the *Free run* option to check if the connection to the control PC was successful and turn on one of the two LED lights on the warehouse
    - *I/O Devices* → *Device (EtherCAT)* → *Term1 (EK1200)* → *Term3 (EL2809)* → *Channel 8* → *Output* → *Online* → *Value = 1*
- Check if the configuration was successful
  - *Check configuration*
  - *Activate configuration*
  - *Run mode*
  - Status bar in the bottom right corner of the TwinCAT System Manager window turns green.

Assignment 2: Warehouse Control Program

In this assignment you will use TwinCAT PLC Control development environment to write a program for warehouse control. Warehouse control program will be running in 2 modes of operation:

- loading a package to the warehouse (the program needs to know the state of the warehouse - which locations are free)
- unloading a package from the warehouse (the program needs to know which package to unload)

I/O Variables

First, input and output variables for control of warehouse sensors and actuators will be created in TwinCAT PLC Project. Variables must be declared as **global**.

Declare 15 input variables and set them to link to the addresses of matching sensors on the warehouse, i.e. digital input terminal EL1809 physical connections.

Also declare 8 output variables and set them to link to the addresses of matching actuators on the warehouse, i.e. digital output terminal EL809 physical connections.

Use an appropriate Structured Text syntax for variables declaration:

- `variable_name AT \%IXaa.a : BOOL` - for input variables
- `variable_name AT \%QXaa.a : BOOL` - for output variables

I/O variables are declared as boolean variables because the controlled sensors and actuators work in a binary mode (on / off or true / false).

`aa.a` are addresses on the input and output terminals and are automatically assigned during the I/O configuration. Check the addresses in the I/O Configuration section of TwinCAT System Manager.

Use tables A.1 and A.2 to help you link and map programming variables to pins on input and output terminals, i.e. functions of the warehouse model. Table A.3 will help you to easier understand where are the various positions of the warehouse tower and the package feeder.

After all I/O variables are correctly declared, rebuild the PLC project and refresh PLC Configuration in System Manager. Additional options appear in PLC Project menu. *Inputs* and *Outputs* options contain a list of all declared input and output variables in the PLC Project. These variables need to be linked and mapped to actual physical connections on the input and output terminals.

Linking variables to the correct address is done by:

- right click on a variable → *Change Link* → select the correct connection

Mapping variables is done by:

Terminal	Pin	Address	Variable	Function	Sensor	
					detect	no detect
Term. 2	1			X axis at position 1 (X+)	false	true
EL 1809	2			X axis at position 2	false	true
inputs	3			X axis at position 3 (X-)	false	true
	4			Y axis at position 1 (Y+)	false	true
	5			Y axis at position 2 (middle)	false	true
	6			Y axis at position 3 (Y-)	false	true
	7			Z axis above position 1 (Z+)	false	true
	8			Z axis below position 1	false	true
	9			Z axis above position 2	false	true
	10			Z axis below position 2	false	true
	11			Z axis above position 3	false	true
	12			Z axis below position 3 (Z-)	false	true
	13			feeder full	true	false
	14			control switch 1	true	false
	15			control switch 2	true	false

Table A.1: Warehouse Model Inputs - Mapping from Connected Pins to Program Variables

Terminal	Pin	Address	Variable	Function	Actuator	
					turn on	turn off
Term. 4	1			X axis to X+ position	true	false
EL 2809	2			X axis to X- position	true	false
outputs	3			Y axis to Y+ position	true	false
	4			Y axis to Y- position	true	false
	5			Z axis to Z+ position	true	false
	6			Z axis to Z- position	true	false
	7			green LED light	true	false
	8			red LED light	true	false

Table A.2: Warehouse Model Outputs - Mapping from Connected Pins to Program Variables

- *Mappings → Generate mappings*

Position	
X axis position 1 (X+)	warehouse tower end position on X axis
X axis position 2	warehouse tower middle position on X axis
X axis position 3 (X-)	warehouse tower starting position on X axis
Y axis position 1 (Y-)	feeder outer position (loading / unloading the package)
Y axis position 2	feeder middle position
Y axis position 3 (Y+)	feeder inner position (inside the warehouse)
Z axis position 1 (Z+)	warehouse tower top position on Z axis (3rd floor)
Z axis position 2	warehouse tower middle position on Z axis (2nd floor)
Z axis position 3 (Z-)	warehouse tower bottom position on Z axis (1st floor)

Table A.3: Warehouse Model Positions Definition

Re-check and re-activate project configuration:

- *Check configuration*
- *Activate configuration*
- *Run mode*
- Status bar in the bottom right corner of the TwinCAT System Manager window turns green.

Minimum Working Example

In TwinCAT PLC Control create a minimum working PLC project. Write the code, build it, download the program to the control PC and run it. **Write a program that will turn both LED lights on the warehouse model on.**

Before downloading the program to the control PC, set the Run-time system:

- *Online* → *Choose Run-time system*
- Select the correct control PC and an available run-time system

Test the program:

- *Login*
- *Download program*
- *Run*, *Stop*, *Reset*
- *Logout*

Complete PLC Project for Warehouse Control

In TwinCAT PLC Control write a complete PLC Project to control the warehouse functionalities. Implement 2 modes of operation - loading the package in the warehouse and unloading the package from the warehouse. Mode of operation is selected using two control switches on the warehouse model.

First, declare a set of variables to define the current state of the warehouse - which warehouse locations are full and which are empty. Warehouse has a 3 by 3 sized field of locations so declare 9 boolean variables, one for each of the locations. Declare this variables as global. These variables will be used during the loading of a new package into the warehouse to find the empty location.

Next, declare two variables that will be used during the unloading of the package from the warehouse and will define the package location (one variable for X axis and one variable for Z axis). Declare these two variables as global. These two variables will be set by a remote function call to unload a package.

Warehouse Control Project must be written as a finite state machine and must include the steps described below:

1. When started the program waits for a command that enables one of two modes of warehouse operation. Two control switches on the warehouse model are used for this purpose. These could either be physically pressed on the model or enabled programmatically by setting the input variable to TRUE.

2. Until one of the two modes of operation is enabled, the red LED light is on. Once one of the two modes is enabled, the red light is turned off and the green LED is turned on. When the current mode of operation is finished, the green light is turned off and the red light is turned on.

3. **Loading the package into the warehouse:**

   a) Check the status of the warehouse and find an available location.

   b) Move the feeder out by Y axis to the outside position and wait for 2 seconds for package loading (timer `TON`)

   c) Move the feeder back in to the middle position. Use the magnet sensor to check if the package was actually loaded. If it was not, repeat the previous step (3.b)).

   d) Move the warehouse tower by X axis to the correct position where you want to put the package.

   e) Move the warehouse tower by Z axis to the correct position. **NOTE**: move the warehouse tower **ABOVE** the position where you want to put the package.

   f) Move the feeder in by Y axis to the inside position.

g) Move the warehouse tower **DOWN** by Z axis to the position **BELOW** the position where you want to put the package.

h) Move the feeder back in by Y axis to the middle position.

i) Move the warehouse tower down by Z axis to the starting position.

j) Move the warehouse tower back by X axis to the starting position.

k) Switch to step 1.

4. **Unloading the package from the warehouse:**

a) From the function call read package location parameters (X axis and Z axis).

b) Move the warehouse tower by X axis to the correct position where you want to take the package from.

c) Move the warehouse tower by Z axis to the correct position. **NOTE**: move the warehouse tower **BELOW** the position where you want to take the package from.

d) Move the feeder in by Y axis to the inside position.

e) Move the warehouse tower **UP** by Z axis to the position **ABOVE** the position where you want to take the package from.

f) Move the feeder back in by Y axis to the middle position.

g) Move the warehouse tower down by Z axis to the starting position.

h) Move the warehouse tower back by X axis to the starting position.

i) Move the feeder out by Y axis to the outside position and wait for 2 seconds for package unloading (timer `TON`)

j) Move the feeder back in to the middle position. Use the magnet sensor to check if the package was actually unloaded. If it was not, repeat the previous step (4.i)).

k) Switch to step 1.

General instructions for PLC Project programming:

- Write the program as a finite state machine. Define the states and conditions for state transitions.
- Consider the fact that the PLC Project will run continuously with a pre-defined run cycle (see Chapter 3.4.3). This demands a thoughtful approach to program design.
- At any bigger change to the PLC program, save the PLC project file, re-build the project and refresh the PLC Configuration in TwinCAT System Manager.

- Use the correct states of input variables (sensors) when they are enabled or disabled. **NOTE**: some sensors are TRUE when they are disabled, while most of the sensors are TRUE when they are enabled (see Table A.1).

- Use the correct values for enabling actuators (motors, LED lights...). Actuators are enabled by setting the appropriate variable to TRUE (see Table A.2).

Assignment 3: Warehouse node

The application should define two types of entities, a warehouse and a package and also functions to work with these types of entities.

The application should include the following functions:

- `getWarehouseState`: the current state of the warehouse; for example: loading the package, unloading the package, extending the feeder to accept / release a package etc. (this could be a textual representation of the finite state machine states in TwinCAT that are used to control the warehouse).

- `getPackages`: get information on all the packages present in the warehouse (id, content, location etc.).

- `load`: main function for loading a new package into the warehouse; this activates the loading mode of the warehouse; the function call include the `getNewPackageLocation` function.

- `getNewPackageLocation`: returns an empty location in the warehouse, to be used by the load function when adding a new package.

- `unload`: the main function for unloading a package from the warehouse; this activates the unloading mode of the warehouse; the function call must include package id or it's location in the warehouse.

For communication with the warehouse over a local network use Modbus protocol. Include `node-modbus` library in Node.js and define a Modbus client. Use the `node-modbus.client.tcp.complete` function to define a new Modbus client with the parameters presented in Table A.4.

Parameter	**Value**
host	192.168.1.6
port	502
unitId	1
timeout	2000
autoReconnect	true
reconnectTimeout	15000
logLabel	'ModbusClientTCP'
logLevel	'debug'
logEnabled	false

Table A.4: node-modbus Module TCP Connection Settings

Modbus TCP client of the `node-modbus` module defines all read and write functions standardized in the Modbus protocol (see Table 3.5 in Chapter 3.6). Use:

- `readHoldingRegisters(start, count)`: to read the values of variables in the PLC application, where `start` is the starting addresses of the first register and `count` is the number of registers to be read
- `writeSingleRegister(start, value)` to change values of variables, where `start` is the address of the register and `value` is a value to be written

As described in Chapter 3.6, a local variable can be defined in a general PLC data area, starting from address `0x6000` (hexadecimal) or in a special PLC memory area, starting from address `0x3000` (hexadecimal) (see Table 3.6). If you use the PLC memory area, you have to declare the variable with the `AT` attribute, for example: `variable_name AT %MX0.0 := TRUE;`. This gives you a direct information about the address of the variable. In case of declaring a variable in a general PLC data area, the compiler independently selects the variable's address (thus no `AT` attribute is used in the declaration) and the address depends on the number, the order and the type of all variables declared in a PLC program.

Table A.5 gives some examples on to how to correctly use Modbus TCP functions to address variables in the PLC application. There are a couple of details that require a comment:

- To load or unload a package from the warehouse declare two extra local variables in the PLC application. Do not try to change the PLC I/O variables of the control switches directly.
- The actual address used in the Modbus TCP client function call needs to be written in the decimal form, so a hexadecimal to decimal conversion is needed (`0x3000` converts to `12288` and `0x6000` converts to `24576`)

- A register is a 16 bit data structure. This especially requires an attention when reading holding register(s) and the function results need to be properly parsed.

First example in Table A.5 shows how to check the current state a warehouse is in. A state variable in the PLC application is declared in the general PLC data area and is the first variable to be declared, therefore it's address is `0x6000`. We only read one register as the state variable is of type integer.

Function	**Modbus function**	**Address**	**Function call**
getWarehouseState	readHoldingRegisters	0x6000 = 24576	client.readHoldingRegisters(24576,1)
getStateOfSlots	readHoldingRegisters	0x3000 = 12288	client.readHoldingRegisters(12288,1)
load a package	writeSingleRegister	0x6050 = 24656	client.writeSingleRegister(24656,1)

unload a package	writeSingleRegister	0x6052 = 24658	client.writeSingleRegister(24632,2);
			client.writeSingleRegister(24656,2);
			client.writeSingleRegister(24632,1)

Table A.5: node-modbus Module Examples

The second example shows how to read the current state of slots in the warehouse. In this case, the 9 variables that correspond to 9 warehouse slots are defined in the PLC memory address, therefore the address starts with `0x3000`. These variables are boolean (TRUE / FALSE) and are 1 bit long, so similarly to the first example only one register needs to be read. The return values are 1 if the slot is occupied or 0 if the slot is free.

Loading a package into the warehouse works similarly to checking the warehouse state. A dedicated local variable is set to 1 (or TRUE) which enables the loading mode of the warehouse. The warehouse automatically find the free slot.

Unloading a package from the warehouse requires a bit more work. First, the location of the package that is to be unloaded must be set. In this example two coordinates of the warehouse 3 by 3 storage field are set to 2. Next, a dedicated local variable is set to 1 (or TRUE) which enables the unloading mode of the warehouse.

Assignment 4: Web GUI for warehouse control (optional)

Build a simple web GUI that will enable visual representation of the warehouse state and control functions. The GUI should enable all of the functions developed in Assignment 4. Figure A.2 shows an example of a simple GUI.

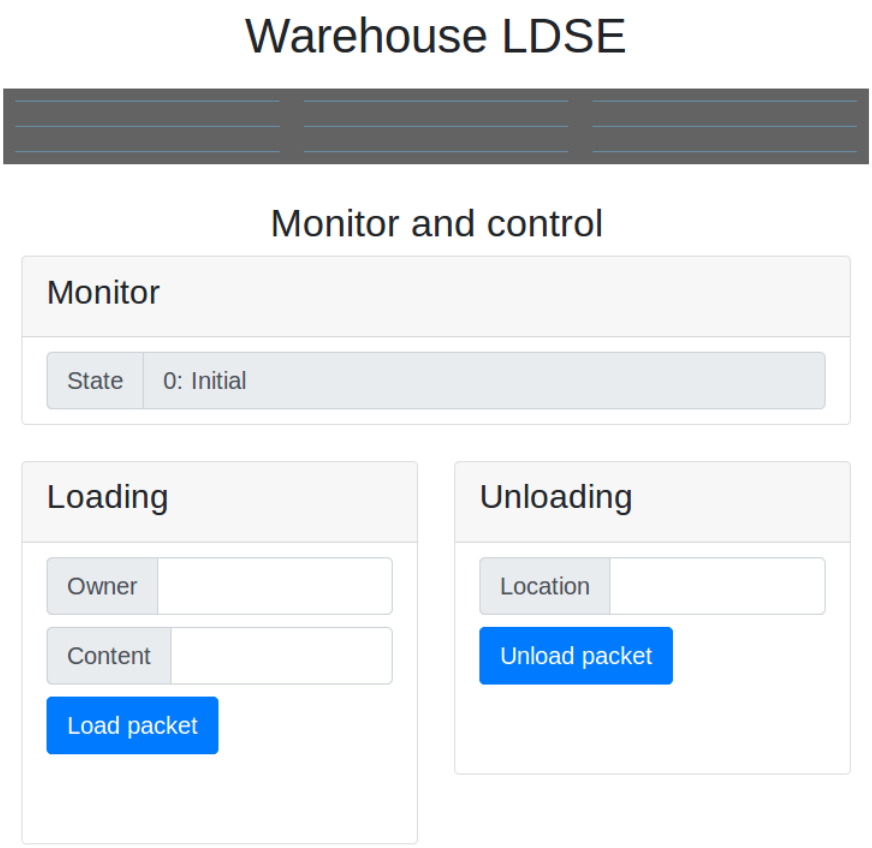

Figure A.2: Example of a Simple Warehouse GUI

A.4 Module 4: Transportation Node

1. Define transportation node properties:

   - location: current location of the transport longitude and latitude coordinates
   - occupancy: storage capacity of the transportation
   - speed: maximum speed of the transport
   - orders: array of collected orders for transport
   - occupation information
   - state:
     parked: transport is parked somewhere and waiting for transportation order
     moving_to_warehouse: transport is moving to the warehouse location to pick up a package
     arrived_at_warehouse: transport has arrived to the warehouse, now the loading of the package can be triggered.
     package_loaded: the package has been loaded to the transport
     moving_to_customer_location: transport is delivering the package to the customer location
     delivered: transport has delivered the package
     moving_to_parked: transport is moving to its parking space

2. Build WEB API endpoints

   (a) `orderTransport`
   Order a new transport for the package and provide all data that is needed to complete the transport. Example: `http://192.168.1.104:3000/orderTransport?buyerlocation='Ljubljana'&id=gfwqgebgwtegvwe&warehouse=192.168.1.103`

   (b) `packageLoaded`
   Warehouse calls this endpoint to confirm that the package has been loaded to the transport and transport can move on. Example: `http://192.168.1.104:3000/packageLoaded`

   (c) `packageDelivered`
   Customer calls this endpoint when the package is delivered to him. Example: `http://192.168.1.104:3000/packageDelivered`

3. Include also economic aspects into the system. The transportation node charges the package a fee for transportation. (optional)

A.5 Integration

Integrate all 4 modules into a fully functional distributed supply chain. Figure A.3 shows an example of the process of ordering a product from our Decentralized supply chain system.

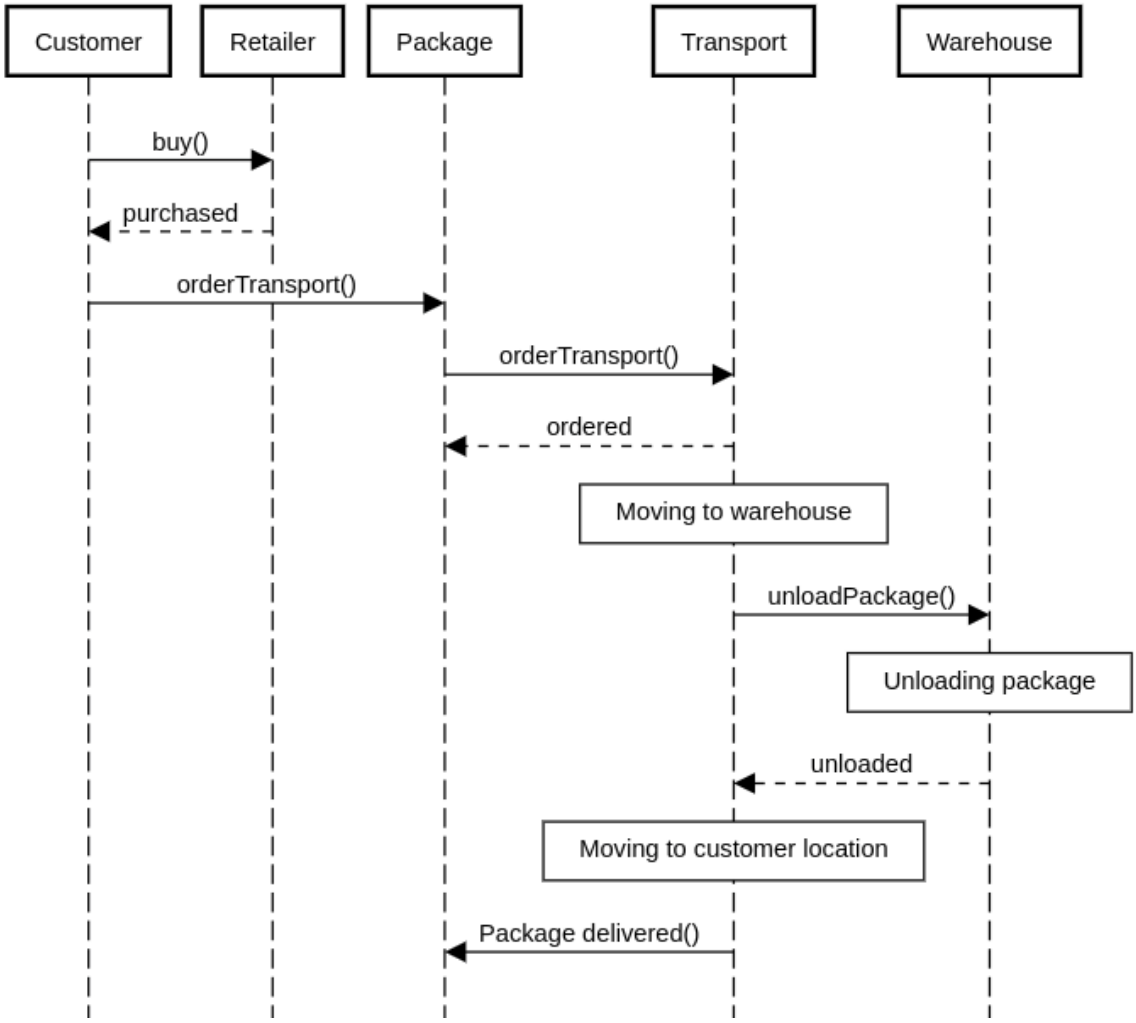

Figure A.3: Example of process of delivering package to the customer

First customer browses the retailer shop for a product. When he decides for one product he buys it by calling web API endpoint `buy()` of the retailer node. Retailer node then responds with the `purchased` response. Customer then calls package endpoint `orderTransport()` which triggers a call of transport endpoint `orderTransport()` from the package node. Transportation node confirms transport by responding `ordered` to the package call. The transportation node is then in the process of moving to the warehouse to load the package when it arrives it calls warehouses endpoint `unloadPackage()`. Warehouse node starts the process of unloading the package which finishes with the response `unloaded` back to the transportation node. Transportation node then starts moving towards customer

location and calls package endpoint `packageDelivered()` when it completes the delivery.

References

[1] P. Zhang. *Industrial Control Technology, A Handbook for Engineers and Researchers*. William Andrew Inc., 2008.

[2] Staudinger - automation website. https://www.staudinger-est.de/en/simulation. Last accessed in May 2018.

[3] Beckhoff automation website. https://www.beckhoff.com. Last accessed in May 2018.

[4] Twincat 2 overview website. https://infosys.beckhoff.com/english.php?content=../content/1033/tcoverview/html/default.htm&id=7054702020850859 55. Last accessed in May 2018.

[5] Twincat 2 system manager overview website. https://infosys.beckhoff.com/english.php?content=../content/1033/tcsystemmanager/basics/tcsysmgr_common_intro.htm&id=1211772295410824515. Last accessed in May 2018.

[6] Twincat 2 plc control overview website. http://www.google.com. Last accessed in May 2018.

[7] Iec 61131-3 standard website. https://webstore.iec.ch/publication/4552. Last accessed in May 2018.

[8] Structured text in beckhoff system website. https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/TcPlcCtrl_Languages\%20ST.htm&id=. Last accessed in May 2018.

[9] Modbus application protocol specification v1.1b3. http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf. Last accessed in May 2018.

[10] Modbus messaging on tcp/ip implementation guide v1.0b. http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf. Last accessed in May 2018.