

Studies in Systems, Decision and Control 476

Marcin Witczak
Lothar Seybold
Eric Bulach
Niko Maucher

Modern IoT Onboarding Platforms for Advanced Applications

A Practitioner's Guide to KIS.ME

OPEN ACCESS

 Springer

Studies in Systems, Decision and Control

Volume 476

Series Editor

Janusz Kacprzyk, Systems Research Institute, Polish Academy of Sciences,
Warsaw, Poland

The series “Studies in Systems, Decision and Control” (SSDC) covers both new developments and advances, as well as the state of the art, in the various areas of broadly perceived systems, decision making and control—quickly, up to date and with a high quality. The intent is to cover the theory, applications, and perspectives on the state of the art and future developments relevant to systems, decision making, control, complex processes and related areas, as embedded in the fields of engineering, computer science, physics, economics, social and life sciences, as well as the paradigms and methodologies behind them. The series contains monographs, textbooks, lecture notes and edited volumes in systems, decision making and control spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the worldwide distribution and exposure which enable both a wide and rapid dissemination of research output.

Indexed by SCOPUS, DBLP, WTI Frankfurt eG, zbMATH, SCImago.

All books published in the series are submitted for consideration in Web of Science.

Marcin Witczak · Lothar Seybold · Eric Bulach ·
Niko Maucher

Modern IoT Onboarding Platforms for Advanced Applications

A Practitioner's Guide to KIS.ME

 Springer

Marcin Witczak
Institute of Control and Computation
Engineering
University of Zielona Góra
Zielona Góra, Poland

Eric Bulach
RAFI GmbH & Co. KG
Berg, Germany

Lothar Seybold
RAFI GmbH & Co. KG
Berg, Germany

Niko Maucher
RAFI GmbH & Co. KG
Berg, Germany



ISSN 2198-4182 ISSN 2198-4190 (electronic)
Studies in Systems, Decision and Control
ISBN 978-3-031-33622-5 ISBN 978-3-031-33623-2 (eBook)
<https://doi.org/10.1007/978-3-031-33623-2>

© The Editor(s) (if applicable) and The Author(s) 2023. This is an Open Access publication.

Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

It always seems impossible until it is done.
Nelson Mandela

Preface

Constant cost pressure forces companies worldwide to work continuously on optimizing internal processes, and hence, improve their overall effectiveness and performance. However, to achieve such a challenging goal, the underlying processes have to be measurable and transparent. This means that irrespective of the considered area, i.e., production or logistics, a set of suitable measures has to be introduced. In production companies, the real-time and automatic recording and displaying of parameters related to quality, performance and availability are of paramount importance. Indeed, they enable optimization in manufacturing processes and automatic calculation of overall equipment effectiveness (OEE) indicators. Another common approach is to employ statistical process control (SPC), which makes it possible to get a predictable behaviour of the manufacturing system and keep its crucial parameters under the constrained control limits. Irrespective of the approach being used, the resulting data can be used, e.g., to measure and analyse downtimes, bottlenecks and other causes of inappropriate performance. As a result, a dedicated implementation of the process optimization and efficiency improvements can be applied.

There is no doubt that digitalization solutions from Industry 4.0 and the Internet of Things (IoT) can be perceived as excellent candidate strategies capable of handling the above stated issues concerning measurements and transparency. However, IoT tools themselves can provide appropriate data only while their efficient integration and application are possible using a dedicated onboarding platform only. To settle this issue, the book undertakes the problem of modern IoT onboarding platforms for the advanced applications pertaining to manufacturing and logistics. In particular, instead of deliberating about a possible hypothetic platforms, an existing and efficient one is employed, which is called KIS.ME. KIS.ME (Keep It Simple. Manage Everything), is a complete IoT solution for a simple integration in manufacturing and logistics. It is composed of a set of hardware devices (KIS.BOX, KIS.IO and KIS.LIGHT), which are intuitively integrated with the cloud platform called KIS.MANAGER. Moreover, the entire platform is an open one, and hence, it enables communication with external services using KIS.API architecture. The application range of KIS.ME is extensive. This is due to the intuitive implementation and visualization of a user-defined KPIs (key performance indicators), which

constitute effective optimization measures. Thus, the potential areas of application of KIS.ME are, e.g., manufacturing, warehouse management and logistics. Indeed, triggering and/or ordering various tasks can be immediately and efficiently implemented with KIS.ME. Such an approach translates directly to the savings of the time and energy.

The book starts with an introductory IoT overview related to its selected scope of applications. Subsequently, a gradual introduction to KIS.ME platform is presented, which constitutes the base for further advanced applications including logistics, control and maintenance of various processes. Finally, the potential of KIS.API communication framework is utilized for an efficient communication with external services. The book contains also training exercises, which gradually introduce the reader into the arcane details of KIS.ME.

Berg, Germany
October 2022

Marcin Witczak
Lothar Seybold
Eric Bulach
Niko Maucher

Contents

1	Introduction	1
1.1	IoT Overview	1
1.1.1	Logistics and Transportation	1
1.1.2	Industrial Applications	4
1.1.3	Agriculture and Environmental Applications	6
1.1.4	Hospitality and Leisure Industry Applications	7
1.1.5	Healthcare	8
1.2	Where Does KIS.ME Go?	9
1.3	Contents of the Book	12
	References	13
2	Onboarding and Preliminary Functionality Training	19
2.1	Preliminaries, Registration and Onboarding	19
2.2	Hierarchical Structure: From Assets and Users to Workspaces	25
2.3	Rights and Permissions	28
2.3.1	User Management	28
2.3.2	User Groups and Workspace Management	32
2.4	Asset Management	33
2.5	Dashboards and Widgets	39
2.6	Digital Twin Design	41
2.7	Datapoints: Plotting and Storing Data	44
2.8	Let Us Go to Workspaces: An Introductory Example with the Floorplan Widget	49
2.9	Let Us Rule: Managing Rules Within a Workspace	54
2.10	State-Space Modelling	58
2.11	Mastering Rule Management: Completeness and Consistency	61
2.11.1	Transforming Conditions	62
2.11.2	Decision Tables	65
2.12	Case Study: Trend Plotting and Performance Analysis	69
2.13	Training Exercises	72

2.14	Concluding Remarks	77
	References	78
3	Towards Logistic Applications	79
3.1	Access Control	79
3.1.1	Managing a Small Warehouse	89
3.2	Two Points–One Transporter	93
3.3	Multiple Points–One Transporter	95
3.4	Multiple Points–Multiple Transporters	105
3.5	Visualizing the Performance of Logistic Applications	109
3.6	Training Exercises	111
3.7	Concluding Remarks	114
	References	115
4	Implementing and Using Essential Statistical Process Control	117
4.1	Data Processing Definitions	117
4.1.1	Calculated Datapoints	120
4.1.2	Key Performance Indicators	125
4.2	Statistical Measures: Location and Variability	132
4.3	Understanding process performance with widgets: A practical way to statistical control charts	135
4.3.1	Single Value Column Chart	140
4.3.2	Single Period and Pie Charts	141
4.3.3	Aggregated Chart	143
4.4	Control Charts: Comparison and Analysis	145
4.4.1	Histograms	146
4.4.2	Control Charts with Limits	149
4.5	Practical Example Revisited	152
4.6	Training Exercises	155
4.7	Concluding Remarks	158
	References	158
5	Mastering System Monitoring and Control	159
5.1	Defining the Performance Cost Function and Its Control	159
5.2	Monitoring the Product Rejection Rate	165
5.3	Demerit System Control	171
5.4	Overall Equipment Effectiveness	177
5.5	Training Exercises	183
5.6	Concluding Remarks	186
	References	187
6	Towards Advanced Applications	189
6.1	Modelling Users and Their Interactions	189
6.1.1	Assembly Process	189
6.2	Transportation Process	194
6.3	Integrating Workers Within a Semi-automatic Assembly System	199

- 6.4 Scheduling Transportation Actions 202
 - 6.4.1 Health-aware and Fault-Tolerant Transportation Scheduling 205
- 6.5 Training Exercise: Work Scheduling 207
- 6.6 Concluding Remarks 208
- References 208
- 7 KIS.API: Towards External Communication 211**
 - 7.1 Introduction to KIS.API 211
 - 7.1.1 User Registration and Authorization 212
 - 7.2 Essential Functionalities 214
 - 7.2.1 Obtaining Information About Asset Groups, Assets and Users 214
 - 7.2.2 Accessing Data Through Datapoints 220
 - 7.2.3 KPIs and Calculated Datapoints 223
 - 7.2.4 Accessing Information About Rules 227
 - 7.2.5 Triggering Rules from External Applications 228
 - 7.3 KIS.API in Practice 229
 - 7.3.1 Feeding MS Excel with KIS.ME Data 229
 - 7.3.2 Feeding Matlab with KIS.ME Data 231
 - 7.4 Triggering Rules from MATLAB 233
 - 7.5 Websockets 234
 - 7.5.1 Brief Introduction to Websockets 234
 - 7.5.2 Obtaining a KIS.ME URI and Identifiers 236
 - 7.5.3 Brief Introduction to STOMP 237
 - 7.5.4 Sample Websocket Implementations 238
 - 7.6 Training Exercises 247
 - 7.7 Concluding Remarks 248
 - References 249
- Appendix A: KIS.ME Commands and Their Sample Applications 251**
- Appendix B: KIS.ME Datapoints and Their Sample Applications 269**
- Appendix C: Glossary 277**
- Index 279**

Acronyms

3C	Communication, computation and control
CDP	Calculated Datapoint
CL	Centre line
GPIO	General-purpose input/output
HMI	Human-machine interface
IIoT	Industrial IoT
IoT	Internet of Things
JIT	Just-in-time
KB	KIS.BOX
KD	KIS.DEVICE
KIS.ME	Keep It Simple. Manage Everything
KL	KIS.LIGHT
KPI	Key performance indicator
LCL	Lower control limit
LO	Linguistic object
MQTT	Message queuing telemetry transport
OEE	Overall equipment effectiveness
PLC	Programmable logic controller
PP	Processing period
RFID	Radio-frequency identification
SCADA	Supervisory control and data acquisition
SOC	State of charge
SOH	State of health
SOP	Standard operating procedure
SSL	Secure socket layer
STOMP	Simple text-oriented messaging protocol
TLS	Transport layer security
UCL	Upper control limit
URI	Uniform resource identifier
URN	Unique resource name

Chapter 1

Introduction



1.1 IoT Overview

The Internet of things (IoT) [1–3] can be formulated as a system of mutually related objects, computing devices, machines, animals, or people, which are equipped with unique identifiers and the possibility of transferring data over a network without the necessity of human-to-human or human-to-computer interactions. Thus, an IoT system consists of connected assets, which can communicate and share information.

The IoT enables assets to observe and interact with the surrounding environment, i.e., they can hear, see, “think” and perform the required actions while sharing information and coordinating decisions. As a result, the IoT transforms the form of these assets from traditional into a smart one. Such a transformation is realized with several important technologies related but not limited to computing, embedded devices, sensor networks, communication strategies as well as Internet protocols. Figure 1.1 illustrates a general IoT structure. The development of the IoT field is persistently stimulated by its application in several domains like the following:

- logistics and transportation [4–6],
- industrial [7–9, 9, 10],
- smart buildings [11],
- agriculture and environment [12–14],
- hospitality and leisure [15, 16],
- healthcare [17].

The objective of the subsequent sections is to briefly review application of the IoT in selected areas. In particular, the review should be perceived as a radar of the main trends rather than a complete state-of-the art analysis.

1.1.1 Logistics and Transportation

The section aims at providing a review of selected works concerning logistics and transportation application. The concept of smart logistics [6] and the related tech-

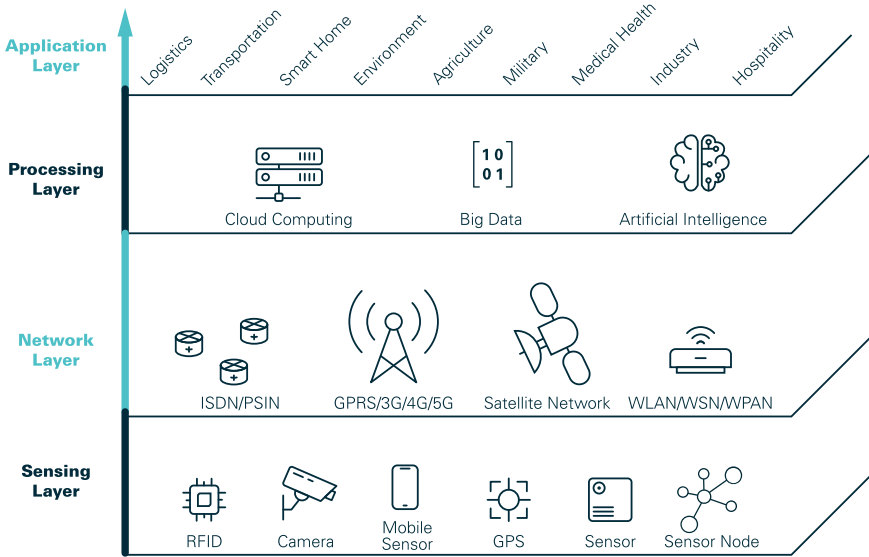


Fig. 1.1 General IoT architecture

nology can be clearly visualized using Fig. 1.2. As in other areas, the number of IoT applications is proliferating constantly. This can be clearly seen in the recent survey papers [4–6]. Nevertheless, there are general application areas, which can be split into the following categories:

- logistics and manufacturing optimization [18–24],
- vehicle status monitoring [25–27],
- cargo status monitoring [28, 29],
- driver monitoring [30–32],
- smart warehousing [33–36].

Thus, the objective of the remaining part of this section is to provide a concise overview of the content of those papers. In particular, in [18], the authors show how to integrate a cloud manufacturing infrastructure with the IoT. As a result, a multi-level dynamic adaptation of a smart logistic synchronization control is attained. The problem of optimizing manufacturing time and energy consumption within a reasonable computing time is proposed in [19]. Such an effect is achieved with an IoT infrastructure coupled with self-organizing configuration mechanisms as well as the related data-driven models. An intelligent IoT-based dispatching platform is proposed in [20]. It couples the IoT with a multi-objective optimization model and a two-level optimization algorithm, which uses the celebrated Dijkstra strategy along with an ant colony approach. A method integrating the blockchain with the IoT is proposed in [21]. The paper shows two real-life examples, which tackle the design and deployment of a logistic and transportation system. In [22], a logistic service supply chain coupled with an IoT-based logistic/service, information and fund flow

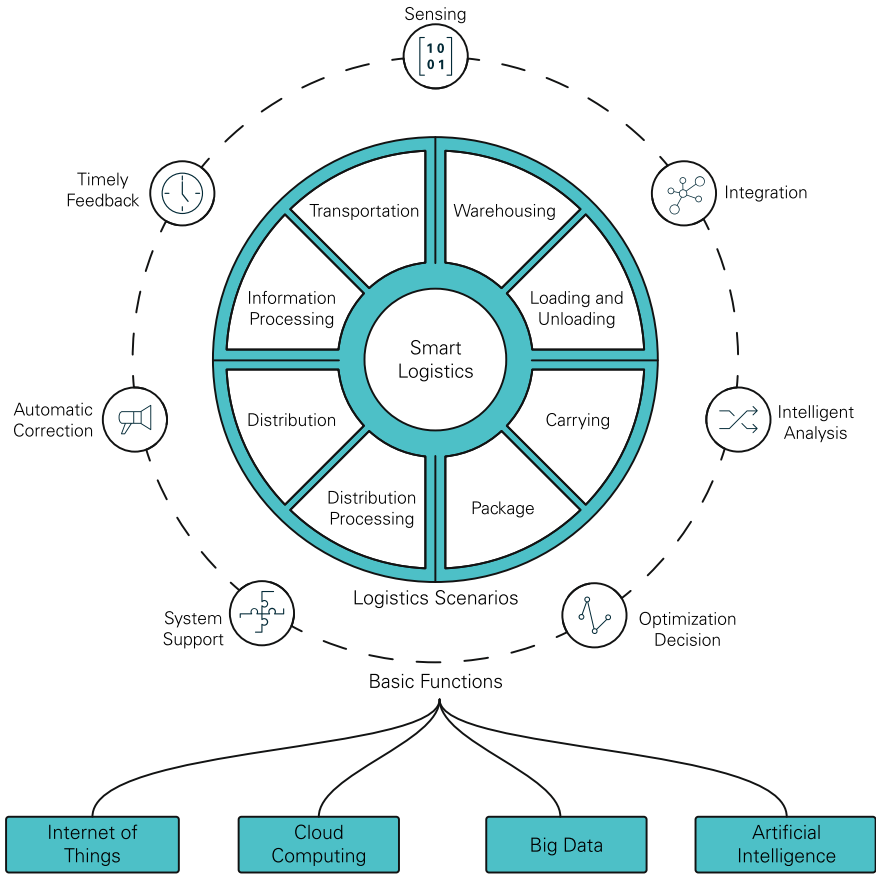


Fig. 1.2 Smart logistics components

is analysed. A decomposition strategy of a complex distribution network is presented in [23]. This is attained by splitting it into smaller nodes, communicating via the IoT. In particular, the IoT forms the basis for a digital twin of the real system, which enables its efficient functioning. An IoT-based real-time sensing model of logistic resources is proposed in [24]. It enables dynamic task distribution within a logistic system.

Several interesting strategies are also proposed in the context of vehicle tracking and monitoring. Indeed, the IoT equipped with a combination of GPS and GSM/GPRS technologies is used to transmit vehicle coordinates and store them in a designated database [25]. The IoT is also employed to vehicle traffic congestion monitoring and control [26]. This strategy allows providing the status and density of the traffic, both of which are transmitted using the Bluetooth technology. Apart from vehicle tracking, a cargo monitoring system is proposed in [28], where cargo tracking is realized with real-time data gathering and suitable information process-

ing. Another IoT-based intelligent cargo tracking system is presented in [27]. It overcomes the unappealing effect of losing GPS signals in environments like dense urban areas and underground tunnels. In particular, a combination of radio-frequency identification and a global system for mobile communication is efficiently used to overcome the difficulties with GPS. In [29], an intelligent cargo solution is proposed, which adapts the IoT to spread information processing between mobile and distributed devices. As a result, they are communicating with themselves as well as with a dedicated platform. The services provided by this platform include cargo localization, rerouting and condition monitoring without human intervention.

Another important aspect pertaining to the performance and safety of logistic and transportation systems is human factors. Indeed, it is very important to monitor driver behaviour in real-time. For that purpose, the work [30] proposes a fusion of the IoT and fog computing. In this architecture, all of the driver's influential, environmental, and vehicle factors are analysed using multiple sensors. In [31], a connected car architecture along with an IoT architecture is used for designing a model for driver behaviour analysis. The strategy is implemented with embedded devices, smartphones as well as a dedicated cloud service. A driver style assessment system is presented in [32]. It uses a dedicated IoT to assess of the driving style using vehicle measurements like speed, acceleration, jerking, engine rotational speed as well as driving time.

The last group of logistic and transportation systems concerns warehouse management as well as related forklift performance. In [33], the authors propose an IoT-based warehouse management system with advanced data analysis and computational intelligence. The system aims at increasing productivity and picking accuracy, efficiency as well as robustness to order variability. The work [34] proposes an IoT infrastructure which combines location information with warehouse working procedures. The system uses a proactive tracking architecture employing the iBeacon tag technology and the concept of distributed gateways. There are also several works dealing with forklifts, e.g., [35, 36]. In the first one [35], a forklift is equipped with an RFID transceiver, a pallet cage as well as electromagnetic field measurement. These can then be used for various analytical deliberations. The second work [36] tackles the problem of improving forklift dispatching and reducing a costs associated with the delays of loading/unloading delivery trucks. For that purpose, forklifts are equipped with sensor nodes that enable collecting such data as the forklifts' physical location, usage time, bumping/collision history, and the battery status. Finally, the information acquired is combined with inventory information and fed into a sophisticated stochastic learning system, which generates dispatching recommendations.

1.1.2 Industrial Applications

As in the case of logistics and transportation, the number of industrial IoT application is also proliferating constantly. This can be clearly seen in the recent survey papers

[7–10, 37]. Generally, industrial applications can be split into the following categories:

- users and system monitoring [38–42],
- programmable logic controllers (PLCs) [43, 44],
- supervisory control and data acquisition (SCADA) [45–49],
- mobile robots [50, 51],
- cyber-physical systems [51–53].

These application areas are rather evident as it is obvious that in any kind of industry one can easily find PLCs and SCADA systems. Thus, in [43], a device called an IoT-PLC is proposed, which can be perceived as a PLC tailored for Industry 4.0. The crucial feature of this device include: regulatory control capabilities, fog-computing functionalities such as filtering and field data storage, and multiple wireless interfaces managed independently. It also uses digital twins of real devices, and hence it can transparently interact with the upper cloud level. As the IIoT can be implemented for critical infrastructures, while attacks on them may cause significant disruptions. In [54], the authors developed a PLC and IoT-based indoor power line communication system. Subsequently, the work [44] investigates an approach for efficient transmission of data between the PLC and cloud platforms.

As SCADA systems are commonly used to control IIoT, the authors of [45] investigate the results of attacking them and provide some general guidelines for security improvement. In [47], a SCADA system which is based on an open source Thingier platform is proposed. It incorporates IoT-oriented web services with the conventional SCADA. The IoT infrastructure includes several sensors, e.g., current and voltage ones, which are used for control purposes. In [48], the authors investigate security and privacy of SCADA-based IoT critical infrastructures. In particular, a dedicate toolbox is proposed, which can tackle the above issues in an efficient way. The toolbox incorporates such functionalities as cryptography-based access to cloud services using identity-based cryptography and signature schemes at the fog layer. A SCADA system which integrates the IoT technology for real-time water quality monitoring is proposed in [49]. The developed system can determine the contamination of water, leakage in pipelines, and some crucial water parameters, e.g., temperature, flow, color, etc. Subsequently, an IoT platform for real-time production performance analysis and exception diagnosis is proposed in [55]. As a result, a decision tree is used for diagnosing exceptions and providing concise information about them.

Another industrial application concerns the development of widely understood cyber-physical systems. In particular, in [56], the authors proposed a platform and software architecture describing features like semantic device interoperability and entity virtualization. This IoT-oriented strategy allows locating and selecting available resources and devices. A wireless sensor network-based process automation monitoring architecture is proposed in [41]. In the approach, the monitoring information is shared in accordance with widely used management standards. In [52], a cyber-physical system which makes it possible to virtualize manufacturing resources is proposed. In the work [57], the authors present an IoT architecture which translates unique identifiers of physical objects to concrete network addresses. As a result,

information about an object, e.g. its status, location, etc., can be extracted. An IoT architecture based on the OPC.NET specification which can be employed for both industrial applications and smart buildings is proposed in [58]. A cyber-physical architecture for the Industry 4.0-based power equipment detection system is proposed in [51]. It integrates many kinds of technologies, including virtual instrumentation, detection and measurement, mechanical and electrical devices, network communication and mobile clients. As a result, the cyber-physical system provides coordination among networks and physical layers. This should be realized by a suitable symbiosis of computation, communications and control (3C) technologies. The resulting 3C framework can perform real-time sensing, control and diagnosis of complex industrial systems [59].

Finally, an IoT system capable of localizing mobile robots is proposed in [50]. It employs neural networks (see, e.g., [60, 61]) for image processing purposes. The resulting mechanism uses the topological mapping method to gain orientation in the explored environment.

1.1.3 Agriculture and Environmental Applications

The number of agriculture-oriented applications is also constantly increasing, and hence their optimization with respect to various factors, e.g., water consumption or soil quality, is of paramount importance. Apart from agriculture, we are currently witnessing a significant development of green energy sources. In both, these fields, the IoT technologies [12–14] play crucial roles in fostering their further development. Indeed, the development areas can be roughly characterized of:

- energy management [62, 63],
- air monitoring [64, 65],
- soil monitoring [66],
- water monitoring [67, 68],
- plant monitoring [69],
- solar panels [62, 70],
- wind turbines [71–73].

In [74], the authors proposed solar tracking system, which eliminates the unappealing shading effect of nearby solar panels, which is encountered in instant positioning of solar panels. They tackle this problem through the use of an IoT-based solution deployed in solar panels, which increases solar energy harvesting efficiency. The work [62] proposes a new energy management strategy for solar-powered devices that intend to power the load directly from the solar cell. This strategy avoids converting and storing the energy, which eliminates losses. Concerning wind turbines, the work [71] uses the IoT for monitoring their behaviour. The authors of [72] analyse recent trends in the application of the IoT in energy generation, specifically in relation to wind energy generation. They investigate potential uses of the IoT pertaining to its

integration with energy generation systems, monitoring and control, as well as maintenance and prediction. The interesting work [73] proposes an 8×8 (64) IoT-based wind farm platform which is built using miniaturized wind turbines with wireless connectivity. Such an IoT grid can measure wind speed and wind direction, which is necessary for optimal wind farm management. The work highlights the potential of using an inexpensive wireless, battery-powered IoT rather than a data-logger that has limited data storage and cannot be accessed remotely.

Concerning agricultural applications, an intelligent approach for efficient plant irrigation is proposed in [70]. It utilizes the IoT and a set of sensors for recording plants data as well as their watering requirements. The system is implemented with a mobile phone, and hence it allows continuous monitoring and control of irrigation efficiency. The work [63] proposes a framework for green mobile crowd sensing. The approach is designed in such a way that only a selected set of best performing sensors is used, which allows significant energy savings. An IoT-based real-time microclimate monitoring system is proposed in [64]. The system includes temperature and relative humidity sensors, powered by solar panels. An environmental monitoring system for apple orchards is presented in [65]. Subsequently, a framework for monitoring multi-layer soil temperature and moisture is proposed in [66]. The framework consists of monitoring nodes, a gateway node and a system platform. The authors of [67] propose an IoT solution for water quality assessment through the measurement of conductivity, temperature, and turbidity. In [68], an IoT infrastructure is presented which consists of sensor nodes providing hydrographic information. Finally, an IoT-based water irrigation scheduling platform is proposed in [69]. In particular, the authors develop a decision support system for irrigation scheduling of olive fields.

1.1.4 Hospitality and Leisure Industry Applications

Since the hospitality and leisure industry plays an important role in our lives, the IoT technology also contributes actively to their permanent development [15, 16]. The main trends can be summarized as follows (cf. Fig. 1.3):

- automatic check-in [75, 76],
- guest experience [16, 77, 78],
- cashless payment systems [79],
- security, privacy and ethical issues [80].

The paper [75] proposes a location recognition algorithm for automatic check-in applications. It can be implemented with smartphones and integrated with a designated cloud platform. The system uses GPS and WiFi access points, which results in a new strategy called a WiFi fingerprint. In [76], the authors propose a malicious check-in defense scheme. They also use the concept of the WiFi fingerprint and make it secure against in unpermitted access. Another improvement of the scheme proposed in [75] is given [81]. The contribution enhances the existing strategy and

Fig. 1.3 Trends in the hospitality and leisure industry



eliminates the need for using GPS. Another automatic vehicle check-in check-out strategy is proposed in [82]. It eliminates the manual entry process and minimizes the security personal effort by exploiting the image data of the vehicle number plate. The work [77] highlights the application of a wristband (Disney’s MagicBand) that serves in Disney World as a credit card, FastPass, hotel key, etc. The authors investigate the IoT impact on enhancing guest experience as well as to better understand guest behaviour and needs. Another IoT cashless payment system for the hospitality industry is proposed in [79]. Subsequently, in [78], the authors explore the influence of demographic factors (education, gender, age, etc.) on consumer attitudes and intentions for using IoT in the hospitality industry. The work [83] proposes an IoT architecture for the hospitality industry located in the so-called smart cities. Finally, an IoT-based authentication system for a remote opening and closing door lock is presented in [84].

1.1.5 Healthcare

Similarly as in the hospitality and leisure industry, the number of IoT applications in healthcare is proliferating [17]. Generally, the applications can be split into two groups:

- health parameters [85, 86],
- patient behaviour [87–90].

In particular, the work [85] proposes a new signal quality-aware IoT electrocardiogram telemetry system for continuous cardiac health monitoring. Similarly, IoT electrocardiogram telemetry is used in [91, 92] analysis and classification pertaining to heartbeat diagnosis. An IoT-based e-health monitoring system is proposed in [93]. An appealing property of this approach is that it aims at controlling temperature raising caused by an on-body sensor, which affects skin comfort. An IoT system monitoring cardiac arrhythmia is presented in [86]. Moreover, an IoT-based heart rate monitoring system is proposed in [94].

Concerning patient behaviour, the authors of [87] propose an IoT device in the form of wearable smart glasses, which are able to monitor eye blinks. In [88], a healthcare digital twin of a patient is presented which utilizes the IoT technologies and AI models to diagnose the state of health and provide a set of clinical questions leading to the final diagnosis. The authors of [89] employ a foot movement monitoring strategy for detecting an early stage of the Alzheimer disease. Finally, the work [90] proposes an IoT system which aims at enhancing speech-language capabilities of patients with Parkinson's disease.

1.2 Where Does KIS.ME Go?

Permanent cost pressure rises the need for a continuous optimization of internal processes, and hence, improve their overall effectiveness and performance. To attain such a challenging objective, the underlying processes have to be measurable and transparent. This means that irrespective of the considered application area, a set of suitable measures has to be introduced. Generally, the real-time and automatic recording and displaying of parameters related to quality, performance and availability are of paramount importance. Indeed, they enable optimization in manufacturing processes and automatic calculation of OEE (overall equipment effectiveness) indicators. Another common approach is to employ SPC (statistical process control), which makes it possible to get a predictable behaviour of the manufacturing system and keep its crucial parameters under the constrained control limits. Irrespective of the approach being used, the resulting data can be used, e.g., to measure and analyse downtimes, bottlenecks, and other causes of inappropriate performance. As a result, a dedicated implementation of the process optimization and efficiency improvements can be applied.

There is no doubt that digitalization solutions from Industry 4.0 and the Internet of Things (IoT) can be perceived as excellent candidate strategies capable of handling the above stated issues concerning measurements and transparency. Such solutions should be applicable for both humans and the machines. Even more, they should integrate them to make their cooperation as efficient as possible.

Therefore, instead of deliberating about hypothetical IoT onboarding platforms capable of handling the above stated challenges, an existing and efficient one is employed within the framework of this book. The onboarding platform is called

Fig. 1.4 KIS.ME:
Communicate, develop,
deploy, control



KIS.ME and the remaining part of this section answers all crucial questions pertaining to its practical advanced applications.

KIS.ME (Keep It Simple.Manage Everything) is an IoT platform which was designed under the light of the following sentence:

*Digitization made easy:
It can be this easy to optimize your process and increase efficiency.*

The justification of this sentence can be split into four crucial components (Fig. 1.4):

- communicate,
- develop,
- deploy,
- control.

Let us start by stating fundamental questions concerning communication:

1. How to communicate with human operators?
2. How to communicate with machines?

To answer these, it is assumed that the processes considered are discrete- event ones [95, 96]. To make the discussion clearer, let us provide the definition of the system state, which is a set of variables that can be used to describe the system's past and future behaviour. Thus, the discrete event system is a discrete-state, event-driven one. This means that its state can only take discrete values from a possibly infinite set. Moreover, its state evolution depends solely on the occurrence of discrete events over time. In the IoT framework, events can be generated using both IoT devices and the cloud platform connected with them. Under these preliminaries, one can quickly figure out that buttons and lights are the most common communication tools used by human operators. Contrarily, machines can be communicated through various inputs and outputs located at their inlets and outlets, respectively. Thus, by finding a common denominator between humans and machines, KIS.ME offers three devices:

- KIS.BOX: a two-button box with digital inputs/outputs,
- KIS.LIGHT: a signal lamp with digital inputs/outputs.
- KIS.IO: an input/output communication box.

These KIS.Devices are communicated through WiFi and should be perceived as a universal hardware inherently integrated with the cloud application called KIS.MANAGER. Note that both devices can communicate discrete states through both digital inputs/outputs or by illuminating colors with the buttons and a lamp.

Let us proceed to crucial questions pertaining to the development stage:

1. How to digitize existing or new processes in a possibly fast and effective way?
2. How to manage and interfere state transitions?
3. How to assess system behaviour and visualize its performance?
4. How to assess system effectiveness and performance predictability?

The answer to the first question is associated with preparing a KIS.Device installation scheme, i.e., it should be located in such a way as to provide appropriate transition of the discrete state. This can be realized through colored lights or digital inputs/outputs of KIS.Devices. The second question is answered by the functionalities of KIS.MANAGER. In particular, it can process the data gathered from KIS.Devices and interact with them through the so-called Rule engine, which should be perceived as a rule base shaping system behaviour. As for third question, it is necessary to explain that the data from KIS.Devices is represented in KIS.MANAGER with the so-called Datapoint. Each numerical or logical Datapoints can be visualized, processed or analysed within a given period. As a result, key performance indicators (KPIs) can be intuitively formulated using a predefined list of commands. Moreover, system transparency can be settled with digital twins of KIS.Devices that can be obtained in KIS.MANAGER. Additionally, digital twins can be located within the floorplan, which is a virtual counterpart of the real system structure. To answer the last question, appropriate KPIs and performance cost functions can be defined. They can be used for the calculation of overall equipment efficiency (OEE) as well as to form control charts to be used for observing and analysing system predictability. Indeed, these control charts form the basis for statistical process control (SPC), which is a commonly used tool for assessing system performance and predictability.

Usually the deployment stage can be a bottleneck on the way towards better performing systems. Indeed, it frequently requires integration of hardware and software provided by different manufacturers. As a result, a third party integrator is usually needed, to make it possible to complete the deployment stage. KIS.ME provides an integrated hardware/software platform, and hence the deployment stage is significantly simplified and does not require extraordinary efforts or costs. In other words, an optimally performed development stage yields smooth realization of the deployment one.

The control stage addresses the following questions:

1. What is process availability?
2. What is process performance?
3. What is process quality?
4. Is the process in the statistical control state?
5. What are (if any) special cause process variations?

KIS.ME provides all necessary and sufficient tools for assessing individual or combined processes' availability. This can be achieved with KPIs calculating an exact run time of the associated equipment. Subsequently, it can be compared with the planned time. Similarly, knowing an ideal process cycle time, a KPI can be implemented calculating the total number of process cycles. Finally, by multiplying it by the ideal process cycle time and then comparing the result with the run time one can obtain process performance. Process quality can be measured in a binary way or with multi-valued quality levels. In both cases KIS.ME can provide a suitable set of KPIs capable of assessing process quality in an intuitive and efficient way.

To answer the last question we should clarify when the process is in the statistical control state. Namely, the process is in the state of control if it is subject to common cause variations only, which can be expected in any set of observations. These common cause variations are predictable and limited, e.g., the discrepancy between consecutive battery mounting times. Indeed, it varies in time but it is possible to assess the range of its variability. Contrarily, special cause variations can be associated with unexpected and unappealing factors that impair process realization, e.g., an equipment fault, quality issues, human operator-related issues, etc. As a result, it is said that the process is in the out-of-control state if it is subject to both common and special cause variations. Thus, apart from the inevitable random and typical fluctuations, other unappealing factors affect process parameters. All these factors can be communicated to KIS.MANGER using KIS.Devices.

Although KIS.ME was originally intended for logistic and industrial applications, it can be applied to settle various tasks encountered in Industry 4.0 and beyond. Thus, the objective of the subsequent part of this book is to introduce the reader into the arcane details of KIS.ME.

1.3 Contents of the Book

The remainder of the book is divided into six chapters and two appendices. However, the book can be generally divided into two parts. The first one introduces the KIS.ME platform as a modern IoT onboarding one. The second one discusses theoretical and practical aspects of applying such a platform for a wide spectrum of advanced applications ranging from logistics and SPC to advanced process and transportation scheduling algorithms. In particular, Chap. 2 introduces the reader into the KIS.ME IoT platform and its hierarchical overview. It discusses also an intuitive concept of Datapoints, which constitute the main source of knowledge about the current status of an asset. It is also shown how to place assets inside workspaces and associate with them a set of functional rules using Rule engine. In Chap. 3, a set of practical guidelines for implementing various logistics-oriented applications is presented. This part starts with a crucial issue pertaining to access control using KIS.Devices and external RFID readers. The remaining three sections deal with transportation systems as well as their digitization and visualization. Subsequently, Chap. 4 aims at introducing the concepts of calculated Datapoints and key performance indicators,

which form the basis of statistical process control. Having such tools, a set of widget charts is introduced, which enable data visualization and analysis. Subsequently, practical examples concerning the development of statistical control charts are presented. Chapter 5 aims at exploiting the methods and tools described in the preceding parts to develop a set of selected process monitoring and control schemes. In particular, it starts with the transportation system, which operates on a set of selected routes. To measure the performance of such a system, a suitable cost function is introduced, which can be monitored and controlled by the designated supervisor. Subsequently, quality control strategies are proposed and described in detail. The last process monitoring strategy aims at calculating and visualizing overall equipment efficiency, which is widely perceived as a key measurement tool for assessing both productivity and efficiency. The objective of Chap. 6 is to provide a list of selected potential applications of KIS.ME. They stem from scientific deliberations of the authors and can be perceived as prospective proofs-of-concept, which can be deployed in various industries. The chapter starts with modelling users and their interactions, which include various important components, e.g., experience and performance. The resulting models make it possible to schedule the work of human operators in a reasonable and predictable way. The chapter discusses also the issues of health-aware and fault-tolerant control and shows a general solution which can overcome these important problems. Subsequently, the objective of Chap. 7 is to provide a concise overview of KIS.API, which can be used for an effective communication with external applications.

All chapters are summarized with a set of exercises motivating the reader to obtain further skills pertaining to practical applications of the modern IoT.

Finally, Appendix A provides a list of KIS.ME commands along with their sample applications. Similarly, Appendix B surveys KIS.ME Datapoints along with their example applications.

References

1. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: a survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutorials* **17**(4), 2347–2376 (2015)
2. D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, J. Henry, *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things* (Cisco Press, Indianapolis, 2017)
3. R. Lohiya, A. Thakkar, Application Domains, Evaluation Datasets, and Research Challenges of IoT: A Systematic Review. *IEEE Internet of Things J.* **8**(11), 8774–8798 (2020)
4. H. Golpîra, S.A.R. Khan, S. Safaeipour, A review of logistics internet-of-things: current trends and scope for future research. *J. Indus. Inform. Integr.* **22**(6), 100194 (2021)
5. J. Wang, Z. Yang, Z. Wang. Intelligent logistics cost control based on 5G network and IoT hardware system, in *Microprocessors and Microsystems* (2020), p. 103476 (in Press)
6. Y. Song, F.R. Yu, L. Zhou, X. Yang, Z. He, Applications of the internet of things (IoT) in smart logistics: a comprehensive survey. *IEEE Internet of Things J.* **8**(6), 4250–4274 (2020)

7. M. Younan, E.H. Houssein, M. Elhoseny, A.A. Ali, Challenges and recommended technologies for the industrial internet of things: a comprehensive review. *Measurement* **151**(2), 107198 (2020)
8. D.G.S. Pivoto, L.F.F. de Almeida, R. da Rosa Righi, J.J.P.C. Rodrigues, A.B. Lugli, and A.M. Alberti. Cyber-physical systems architectures for industrial internet of things applications in Industry 4.0: a literature review. *J. Manuf. Syst.* **58**(1), 176–192 (2021)
9. Q. Wang, X. Zhu, Y. Ni, L. Gu, H. Zhu, Blockchain for the iot and industrial IoT: a review. *Internet of Things* **10**(6), 100081 (2020)
10. Y. Liao, E.F.R. Loures, F. Deschamps, Industrial internet of things: a systematic literature review and insights. *IEEE Internet of Things J.* **5**(6), 4515–4525 (2018)
11. A. Verma, S. Prakash, V. Srivastava, A. Kumar, SCh. Mukhopadhyay, Sensing, controlling, and IoT infrastructure in smart building: a review. *IEEE Sens. J.* **19**(20), 9036–9046 (2019)
12. O. Elijah, T.A. Rahman, I. Orikumhi, Ch.Y. Leow, M.N. Hindia, An overview of Internet of Things (IoT) and data analytics in agriculture: benefits and challenges. *IEEE Internet of Things J.* **5**(5), 3758–3773 (2018)
13. C. Brewster, I. Roussaki, N. Kalatzis, K. Doolin, K. Ellis, IoT in agriculture: designing a europe-wide large-scale pilot. *IEEE Commun. Mag.* **55**(9), 26–33 (2017)
14. G. Bedi, G.K. Venayagamoorthy, R. Singh, R.R. Brooks, K.C. Wang, Review of internet of things (IoT) in electric power and energy systems. *IEEE Internet of Things J.* **5**(2), 847–870 (2018)
15. P. Kansakar, A. Munir, N. Shabani, Technology in the hospitality industry: prospects and challenges. *IEEE Consum. Electron. Mag.* **8**(3), 60–65 (2019)
16. J.R. Chang, M.Y. Chen, L.S. Chen, S.C. Tseng, Why customers don't revisit in tourism and hospitality industry? *IEEE Access* **7**(10), 146588–146606 (2019)
17. M.H. Kashani, M. Madanipour, M. Nikravan, P. Asghari, E. Mahdipour, A systematic review of iot in healthcare: applications, techniques, and trends. *J. Netw. Comput. Appl.* **192**(10), 103164 (2021)
18. T. Qu, S.P. Lei, Z.Z. Wang, D.X. Nie, X. Chen, G.Q. Huang, IoT-based real-time production logistics synchronization system under smart cloud manufacturing. *Int. J. Adv. Manuf. Technol.* **84**(5), 147–164 (2016)
19. Y. Zhang, Z. Guo, J. Lv, Y. Liu, A framework for smart production-logistics systems based on CPS and industrial IoT. *IEEE Trans. Indus. Inform.* **14**(9), 4019–4032 (2018)
20. W. Jianxin, K.L. Ming, Z. Yuanzhu, W. XiaoFeng, An intelligent logistics service system for enhancing dispatching operations in an IoT environment. *Transp. Res. Part E: Logist. Transp. Rev.* **135**(3), 101886 (2020)
21. M. Humayun, N.Z. Jhanjhi, B. Hamid, G. Ahmed, Emerging smart logistics and transportation using IoT and blockchain. *IEEE Internet of Things Mag.* **3**(2), 58–62 (2020)
22. W. Liu, Z. Gao, Study on IoT based architecture of logistics service supply. *Int. J. Grid Distrib. Comput.* **7**(1), 169–178 (2014)
23. R. Nejc, V. Rok, C. Marko, P. Tomaz, D. Janez, Distributed logistics platform based on blockchain and IoT. *Procedia CIRP* **81**, 826–831 (2019)
24. L. Sichao, Z. Geng, W. Lihui, IoT-enabled dynamic optimisation for sustainable reverse logistics. *Procedia CIRP* **69**, 662–667, in *25th CIRP Life Cycle Engineering (LCE) Conference*, 30 April–2 May 2018 (Denmark, Copenhagen, 2018)
25. S. Lee, G. Tewolde, J. Kwon. Design and implementation of vehicle tracking system using GPS/GSM/GPRS technology and smartphone application, in *IEEE World Forum on Internet of Things (WF-IoT)*, Seoul, Korea, 6–8 Mar 2014 (IEEE, 2014), pp. 353–358
26. S. Sukode, S. Gite, Vehicle traffic congestion control & monitoring system in IoT. *Int. J. Appl. Eng. Res.* **10**(8), 19513–19523 (2015)
27. J. Prinsloo, R. Malekian, Accurate vehicle location system using RFID, an internet of things approach. *Sensors* **16**(6), 825 (2016)
28. L. Zhou, C.X. Lou, Intelligent cargo tracking system based on the Internet of Things, in *15th International Conference on Network-Based Information Systems*, Melbourne, Australia (2012), 26–28 Sept 2012, pp. 489–493

29. M. Forcolin, E. Fracasso, F. Tumanischvili, P. Lupieri. EURIDICE—IoT applied to logistics using the intelligent cargo concept, in *17th International Conference on Concurrent Enterprising*, Aachen, Germany (2011), 20–22 June 2011, pp. 1–9
30. M. Aazam, X. Fernando, Fog assisted driver behavior monitoring for intelligent transportation system, in *IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto, Canada (2017), 24–27 Sept 2017, pp. 1–5
31. C. Boja, P. Paul, I. Bogdan, Service architecture for driver behavior analysis in an IoT vehicular environment, in *15th International Conference on INFORMATICS in ECONOMY (IE 2016), Education, Research & Business Technologies*, Cluj-Napoca, Romania (2016), 2–3 June 2016, pp. 2284–7472
32. B. Jachimczyk, D. Dziak, J. Czaplą, P. Damps, W. Kulesza, IoT on-board system for driving style assessment. *Sensors* **18**(4), 1233 (2018)
33. C.K.M. Lee, Y. Lv, K.K.H. Ng, W. Ho, K.L. Choy, Design and application of internet of things-based warehouse management system for smart logistics. *Int. J. Prod. Res.* **56**(8), 2753–2768 (2018)
34. Z. Zhao, M. Zhang, C. Yang, J. Fang, G.Q. Huang, Distributed and collaborative proactive tandem location tracking of vehicle products for warehouse operations. *Comput. Indus. Eng.* **125**(11), 637–648 (2018)
35. J.Fl. Hoefinghoff, A. Jungk, W. Knop, L. Overmeyer. Using 3D field simulation for evaluating uhf rfid systems on forklift trucks. *IEEE Trans. Antennas Propag.* **59**(2), 689–691 (2011)
36. R.M. Estanjini, Y. Lin, K. Li, D. Guo, ICh. Paschalidis, Optimizing warehouse forklift dispatching using a sensor network and stochastic learning. *IEEE Trans. Indus. Inform.* **7**(3), 476–486 (2011)
37. I. Butun, *Industrial IoT* (Springer, Berlin, 2020)
38. M. Witczak, B. Lipiec, M. Mrugalski, L. Seybold, Z. Banaszak. Fuzzy modelling and robust fault-tolerant scheduling of cooperating forklifts, in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE, Glasgow, 2020), pp. 1–10
39. M. Witczak, Lo. Seybold, G. Bocewicz, M. Mrugalski, A. Gola, Z. Banaszak, A fuzzy logic approach to remaining useful life control and scheduling of cooperating forklifts, in *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE, Luxemburg, 2021), pp. 1–8
40. P. Majdzik, M. Witczak, B. Lipiec, Z. Banaszak, Integrated fault-tolerant control of assembly and automated guided vehicle-based transportation layers. *Int. J. Comput. Integr. Manuf.* 1–18 (2021)
41. D. Raposo, A. Rodrigues, S. Sinche, J. Sá Silva, F. Boavida, Industrial IoT monitoring: technologies and architecture proposal. *Sensors* **18**(10) (2018)
42. Zf openmatics. https://www.zf.com/products/en/connectivity/products_52107.html. Accessed 14 Sept 2021
43. J. Mellado, F. Núñez, in Design of an IoT-PLC: a containerized programmable logical controller for the Industry 4.0. *J. Indus. Inform. Integr.* **100250** (2021) (in press)
44. A. Gavlas, J. Zwierzyna, J. Koziorek, Possibilities of transfer process data from plc to cloud platforms based on iot. *IFAC-PapersOnLine* **51**(6), 156–161 (2018)
45. G. Falco, C. Caldera, H. Shrobe, IIoT cybersecurity risk modeling for SCADA systems. *IEEE Internet of Things J.* **5**(6), 4486–4495 (2018)
46. S. Samtani, S. Yu, H. Zhu, M. Patton, J. Matherly, H. Chen, Identifying SCADA systems and their vulnerabilities on the internet of things: a text-mining approach. *IEEE Intel. Syst.* **33**(2), 63–73 (2018)
47. L.O. Aghenta, M.T. Iqbal, Low-cost, open source IoT-based SCADA system design using thinger IO and ESP32 thing. *Electronics* **8**(8), 822 (2019)
48. T. Baker, M. Asim, Á. MacDermott, F. Iqbal, F. Kamoun, B. Shah, O. Alfandi, M. Hammoudeh, A secure fog-based platform for SCADA-based IoT critical infrastructure. *Softw. Pract. Exp.* **50**(5), 503–518 (2020)
49. K. Saravanan, E. Anusuya, R. Kumar, L.H. Son, Real-time water quality monitoring using Internet of things in SCADA. *Environ. Monit. Assess.* **190**(9), 1–16 (2018)

50. C. Dourado, P.P. Suane, Raul da Silva, V. da Nóbrega, A. Barros, A.K. Sangaiah, P. Rebouças Filho, V. de Albuquerque. A new approach for mobile robot localization based on an online iot system. *Future Gener. Comput. Syst.* **100**(11), 859–881 (2019)
51. M. Yu, M. Zhu, G. Chen, J. Li, Z. Zhou, A cyber-physical architecture for Industry 4.0-based power equipments detection system, in *International Conference on Condition Monitoring and Diagnosis (CMD)*, Xi'an, China (2016), 25–28 Sept 2016, pp. 782–785
52. W. Liu, J. Su. A solution of dynamic manufacturing resource aggregation in CPS, in *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, Chongqing, China (2011), 20–22 Aug 2011, vol. 2, pp. 65–71
53. N. Jazdi. Cyber physical systems in the context of industry 4.0, in *IEEE International Conference on Automation, Quality and Testing, Robotics*, Cluj-Napoca, Romania (2014), 22–24 May 2014, pp. 1–4
54. M. Jani, P. Garg, A. Gupta, On the performance of a cooperative PLC-VLC indoor broadcasting system consisting of mobile user nodes for IoT networks. *IEEE Trans. Broadcast.* **67**(1), 289–298 (2021)
55. Y. Zhang, W. Wang, N. Wu, Ch. Qian, Iot-enabled real-time production performance analysis and exception diagnosis model. *IEEE Trans. Autom. Sci. Eng.* **13**(3), 1318–1332 (2016)
56. D. Conzon, P. Brizzi, P. Kasinathan, C. Pastrone, F. Pramudianto, P. Cultrona, Industrial application development exploiting IoT vision and model driven programming, in *18th International Conference on Intelligence in Next Generation Networks*, Paris, France (2015), 17–19 Feb 2015, pp. 168–175
57. K. Bill, A dns architecture for the internet of things: a case study in transport logistics. *Procedia Comput. Sci.* **19**, 594–601 (2013)
58. I. Ungurean, N.C. Gaitan, V.G. Gaitan. An IoT architecture for things from industrial environment, in *10th International Conference on Communications (COMM)*, Bucharest, Romania (2014), 29–31 May 2014, pp. 1–4
59. H. Xu, J. Wu, J. Li, X. Lin. Deep reinforcement learning-based cybertwin architecture for 6G IIoT: an integrated design of control, communication, and computing. *IEEE Internet of Things J.* **8**(22), 16337–16348 (2021)
60. M. Witczak, *Modelling and Estimation Strategies for Fault Diagnosis of Non-linear Systems* (Springer, Berlin, 2007)
61. M. Witczak, *Fault Diagnosis and Fault-Tolerant Control Strategies for Non-linear Systems: Analytical and Soft Computing approaches* (Springer International Publishing, Heidelberg, 2014)
62. Y. Wang, Y. Liu, C. Wang, Z. Li, X. Sheng, H.G. Lee, N. Chang, H. Yang, Storage-less and converter-less photovoltaic energy harvesting with maximum power point tracking for internet of things. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **35**(2), 173–186 (2015)
63. M. Marjanović, L. Skorin-Kapov, K. Pripuzić, A. Antonić, I.P. Žarko, Energy-aware and quality-driven sensor management for green mobile crowd sensing. *J. Network Comput. Appl.* **59**(1), 95–108 (2016)
64. N. Watthanawisuth, A. Tuantranont, T. Kerdcharoen. Microclimate real-time monitoring based on zigbee sensor network, in *IEEE SENSORS Conference*, Christchurch, New Zealand (2009), 25–28 Oct 2009, pp. 1814–1818
65. S. Lu, M. Duan, P. Zhao, Y. Lang, X. Huang, GPRS-based environment monitoring system and its application in apple production, in *IEEE International Conference on Progress in Informatics and Computing*, Shanghai, China (2010), 10–12 Dec 2010, vol. 1, pp. 486–490
66. K. Chen, H. Zhang, T. Wu, J. Hu, C. Zhai, D. Wang, Design of monitoring system for multi-layer soil temperature and moisture based on WSN, in *International Conference on Wireless Communication and Sensor Network*, Wuhan, China (2014), 13–14 Dec 2015, pp. 425–430
67. O. Postolache, M. Pereira, P. Girão, Sensor network for environment monitoring: water quality case study, in *4th Symposium on Environmental Instrumentation Measurement*, Lecce. Italy (2013), 3–4 June 2013, pp. 30–34
68. Y. Xijun, L. Limei, X. Lizhong, The application of wireless sensor network in the irrigation area automatic system, in *International Conference on Networks Security, Wireless Communications and Trusted Computing*, 25–26 Apr 2009, Wuhan, China (2009), vol. 1, pp. 21–24

69. M. A. Fourati, W. Chebbi, A. Kamoun, Development of a web-based weather station for irrigation scheduling, in *3rd IEEE International Colloquium in Information Science and Technology (CIST)*, Tetuan, Morocco (2014), pp. 37–42
70. M.S. Munir, I.S. Bajwa, M.A. Naeem, B. Ramzan, Design and implementation of an iot system for smart energy consumption and smart irrigation in tunnel farming. *Energies* **11**(12), 3427 (2018)
71. M.K. Singla, J. Gupta, P. Nijhawan, S. Ganguli, S.S. Rajest, *Development of an efficient, cheap, and flexible iot-based wind turbine emulator, in Business Intelligence for Enterprise Internet of Things* (Springer, Berlin, 2020), pp.225–231
72. S. Karad, R. Thakur, Efficient monitoring and control of wind energy conversion systems using internet of things (IoT): a comprehensive review. *Environ. Dev. Sustain.* **23**(2), 14197–14214 (2021)
73. B. Srbinovski, G. Conte, A.P. Morrison, P. Leahy, E. Popovici, ECO: an IoT platform for wireless data collection, energy control and optimization of a miniaturized wind turbine cluster: power analysis and battery life estimation of iot platform. in *IEEE International Conference on Industrial Technology (ICIT)*, Toronto, Canada (2017), 22–25 Mar 2017, pp. 412–417
74. M. Yakut, N.B. Erturk, An IoT-based approach for optimal relative positioning of solar panel arrays during backtracking. *Comp. Stand. Interf.* **80**(3), 103568 (2022)
75. I. Bisio, F. Lavagetto, M. Marchese, A. Sciarrone, GPS/HPS-and Wi-Fi fingerprint-based location recognition for check-in applications over smartphones in cloud-based LBSs. *IEEE Trans. Multimed.* **15**(4), 858–869 (2013)
76. W. Li, Z. Su, K. Zhang, A. Benslimane, D. Fang, Defending malicious check-in using big data analysis of indoor positioning system: an access point selection approach. *IEEE Trans. Netw. Sci. Eng.* **7**(4), 2642–2655 (2020)
77. A. Pizam, The Internet of things (IoT): the next challenge to the hospitality industry. *Int. J. Hosp. Manage.* **100**(62), 132–133 (2017)
78. A. Aluri, R. Palakurthi, The influence of demographic factors on consumer attitudes and intentions to use RFID technologies in the US hotel industry. *J. Hosp. Tour. Technol.* **2**(3), 188–203 (2011)
79. A.B. Ozturk, Customer acceptance of cashless payment systems in the hospitality industry. *Int. J. Contemp. Hosp. Manage.* **28**(4), 801–817 (2016)
80. S. Mercan, K. Akkaya, L. Cain, J. Thomas, Security, privacy and ethical concerns of iot implementations in hospitality domain, in *International Conferences on Internet of Things (iThings)*, 02–06 Nov 2020, Rhodes Island, Greece (2020), pp. 198–203
81. I. Bisio, Ch. Garibotto, F. Lavagetto, A. Sciarrone, Outdoor places of interest recognition using WiFi fingerprints. *IEEE Trans. Veh. Technol.* **68**(5), 5076–5086 (2019)
82. S. Tenzin, P. Dorji, B. Subba, T. Tobgay. Smart check-in check-out system for vehicles using automatic number plate recognition, in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kharagpur, India (2020), 1–3 July 2020, pp. 1–6
83. P. Kansakar, A. Munir, N. Shabani, A fog-assisted architecture to support an evolving hospitality industry in smart cities, in *International Conference on Frontiers of Information Technology (FIT)*, Islamabad, Pakistan (2018), 17–19 Dec 2018, pp. 59–64
84. J. Jeong, *A study on the iot based smart door lock system, in Information Science and Applications (ICISA)* (Springer, Berlin, 2016), pp.1307–1318
85. U. Satiya, B. Ramkumar, M. Sabarimalai Manikandan. Real-time signal quality-aware ECG telemetry system for IoT-based health care monitoring. *IEEE Internet of Things J.* **4**(3), 815–823 (2017)
86. E. Moghadas, J. Rezazadeh, R. Farahbakhsh, An IoT patient monitoring based on fog computing and data mining: cardiac arrhythmia use case. *Internet of Things* **11**(9), 100251 (2020)
87. A. Sciarrone, I. Bisio, C. Garibotto, F. Lavagetto, G.H. Staude, A. Knopp, Leveraging IoT wearable technology towards early diagnosis of neurological diseases. *IEEE J. Select. Areas Commun.* **39**(2), 582–592 (2021)

88. J. Zhang, L. Li, G. Lin, D. Fang, Y. Tai, J. Huang, Cyber resilience in healthcare digital twin on lung cancer. *IEEE Access* **8**(10), 201900–201913 (2020)
89. Ra. Varatharajan, G. Manogaran, M. Kumar Priyan, R. Sundarasekar, Wearable sensor devices for early detection of alzheimer disease using dynamic time warping algorithm. *Cluster Comput.* **21**(1), 681–690 (2018)
90. H. Dubey, J.C. Goldberg, M. Abtahi, L. Mahler, K. Mankodiya, Echowear: smartwatch technology for voice and speech treatments of patients with parkinson’s disease, in *Proceedings of the Conference on Wireless Health*, Bethesda Maryland, USA, 14–16 Oct 2015, pp. 1–8
91. D. Azariadi, V. Tsoutsouras, S. Xydis, D. Soudris, ECG signal analysis and arrhythmia detection on IoT wearable medical devices, in *5th International Conference on Modern Circuits and Systems Technologies (MOCASST)*, Thessaloniki, Greece, 12–14 May 2016 (2016), pp. 1–4
92. Z. Yang, Q. Zhou, L. Lei, K. Zheng, W. Xiang, An IoT-cloud based wearable ECG monitoring system for smart healthcare. *J. Med. Syst.* **40**(12), 1–11 (2016)
93. E. Selem, M. Fatehy, S.M. Abd El-Kader, H. Nassar, The (temperature heterogeneity energy) aware routing protocol for iot health application. *IEEE Access* **7**(2), 108957–108968 (2019)
94. V. Goel, S. Srivastava, D. Pandit, D. Tripathi, P. Goel, Heart rate monitoring system using finger tip through IoT. *Int. Res. J. Eng. Technol.* **6**(13), 1–4 (2018)
95. L. Seybold, M. Witczak, P. Majdzik, R. Stetter, Towards robust predictive fault-tolerant control for a battery assembly system. *Int. J. Appl. Math. Comput. Sci.* **25**(4), 849–862 (2015)
96. P. Majdzik, A. Akielaszek-Witczak, L. Seybold, R. Stetter, B. Mrugalska, A fault-tolerant approach to the control of a battery assembly system. *Control Eng. Pract.* **55**(10), 139–148 (2016)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 2

Onboarding and Preliminary Functionality Training



2.1 Preliminaries, Registration and Onboarding

Let us start with introducing the definition of a system, which is a part of the universe that can be affected and/or monitored by KIS.ME. Consequently, KIS.ME can be divided into two layers:

- KIS.Device: hardware capable of performing the desired communication tasks within the system,
- KIS.MANAGER: software being a web platform used to affect and/or monitor the system.

Having the system, it is possible to introduce its components, which are defined as assets. They are physical parts of the system and are exemplified by KIS.Devices.

> Unique resource name (URN)

Each KIS.Device is uniquely identified by the URN number, e.g.,

```
urn:rafi:sbox:9c65f93cbf2d.
```

Once assigned, it cannot be further modified.

Thus, an asset group is simply a set of assets. In the current release of KIS.ME, KIS.Devices are divided into

- KIS.BOX: a communication push-button box (see Fig. 2.1),
- KIS.LIGHT: a communication signal lamp (see Fig. 2.2),
- KIS.IO: an input/output communication box (see Fig. 2.3).

Let us start with defining two kinds of LED lights incorporated within each KIS.Device (see Figs. 2.1 and 2.2):

- Status LED: it exhibits the functional state of a KIS.Device and is defined in Table 2.1;

Fig. 2.1 KIS.BOX

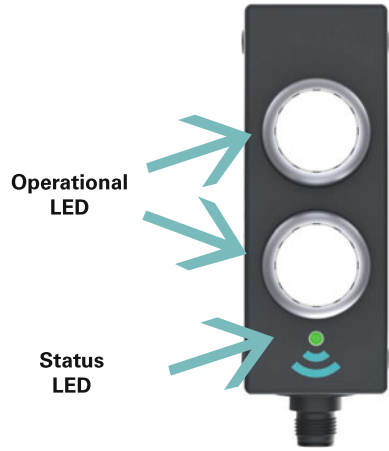


Fig. 2.2 KIS.LIGHT



Fig. 2.3 KIS.IO



Table 2.1 Status LED colors and their meaning

Status LED color	Meaning
Red	Device is booting
Yellow	Device is booted, no WiFi connection
Magenta	Device is connected with WiFi but MQTT Port check Certificate check or NTP time-sync check errors occurred
Blue	Device is connected with WiFi, MQTT Port check and certificate check were successful No connection to KIS.MANAGER
Green	Device is connected to KIS.MANAGER
Turquoise	Update in progress
Flashing Magenta (2Hz)	MQTT Port check failed
Flashing Magenta (1Hz)	NTP time-sync check failed
Flashing Magenta (0.5Hz)	Certificate check failed

Table 2.2 Predefined operational LED colors

Color	RGB HEX code	Integer value
Blue	#0000FF	0
Turquoise	#00FFFF	1
Black	#000000	2
Green	#00FF00	3
Magenta	#FF00FF	4
Red	#FF0000	5
White	#FFFFFF	6
Yellow	#FFFF00	7

Operational LED: it exhibits the individual operational state of a KIS.Device and can be defined by the user using the set of colors provided in Table 2.2.

Note that KIS.IO can be perceived as a simplified version of KIS.BOX without buttons. Subsequently, a general KIS.ME framework overview can be introduced, which is portrayed in Fig. 2.4. Subsequently let us proceed to introduce KIS.Device functionalities. As can be observed in Fig. 2.4, each KIS.BOX can be perceived as a human-machine interface (HMI) with two push-buttons, linked with the KIS.MANAGER via WiFi. Each pushbutton contains an operational LED, which illuminates in an RGB color. However, for the sake of simplicity, the list of colors is limited in KIS.MANAGER to those presented in Table 2.2. Thus, they can be identified by either the RGB HEX code or an integer value. It should be also noted that the black color is used to signify the fact that a corresponding operational LED is not lit. Compared to the KIS.BOX, KIS.LIGHT has limited functionalities as its pri-

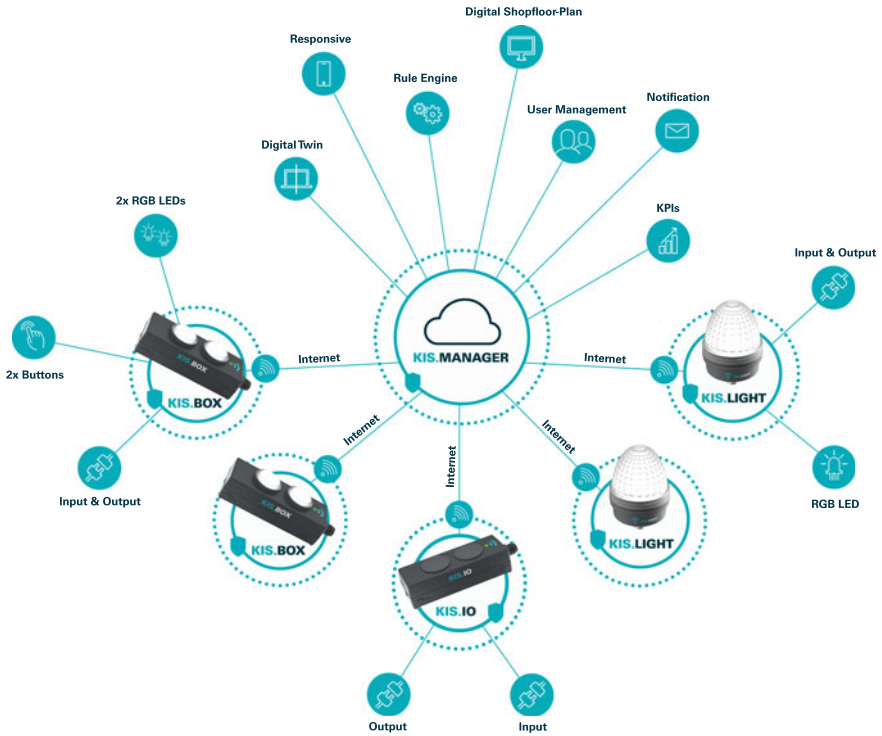


Fig. 2.4 General KIS.ME overview

Table 2.3 KIS.Device parameters

Name	Value
Luminous element color	RGB
Degree of protection	IP65
WLAN standard	IEEE 802.11 b/g/n 2.4 GHz
Connection terminal	M12 8-pin A-coded
Operating voltage	5 ± 10% V, 24 ± 20% V
GPIO	2 inputs/ 2 outputs

Table 2.4 M12 PIN specification

PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8
VCC voltage	In 1	GND	In 2	Out 1	Out 2	USB D+	USB D-

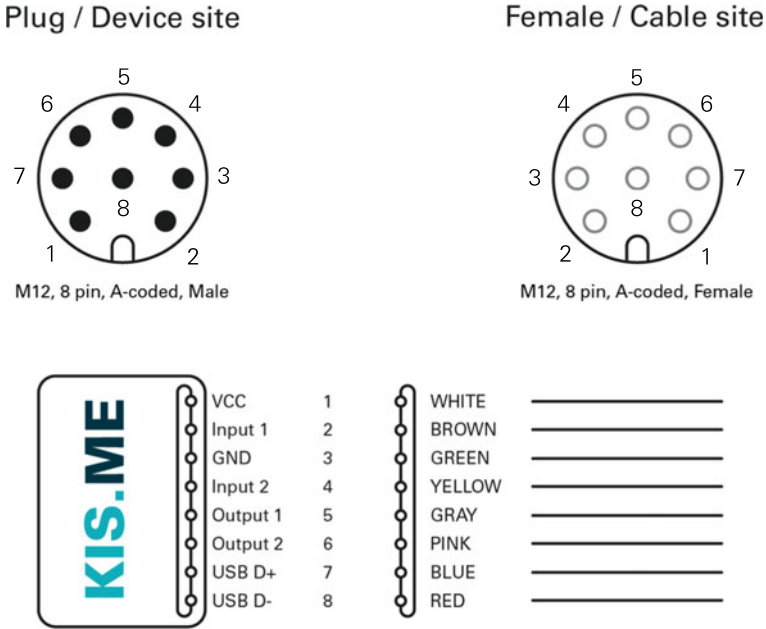


Fig. 2.5 KIS.Device M12 connections

mary purpose is to be a signalling lamp with one operational LED only. Finally, the essential technical parameters of KIS.Devices are given in Table 2.3. It can be noted that each KIS.Device is fed with an M12 8-pin A-coded connection terminal, while the purpose of particular PINs is given in Table 2.4. The operating voltages can be either 5 V or 24 V. In the first case, a KIS.Devices are fed through USB, while in the second one, the respective 24 V voltage is fed through PINs 1 and 3. Each KIS.Device possesses a general purpose input output (GPIO) interface, which is available while using a 24 V power supply only. As can be observed in Table 2.4, GPIO is composed of two digital inputs and two outputs. Thus, each KIS.Device enables attaining the functionalities portrayed in Fig. 2.5, which can be further exploited in a large number of practical applications. Finally, it should be noted that a full technical documentation of KIS.Devices is available at <https://kisme.rafi.de/en/>. Having the hardware layer provided, let us proceed to the software one. As can be observed in Fig. 2.4, the main KIS.MANAGER features can be summarized as follows:

- System assets can be easily digitalized through digital twins of KIS.Devices.
- The system is divided into workspaces, i.e., selected parts of the system, which inherit its desired set of assets.
- A graphical representation of the workspaces can be introduced using floorplans.
- An asset’s behaviour can be affected and monitored using the Rule engine functionality, which makes it possible to implement functional IF-THEN rules governing interactions between assets.

- The behaviour of the system and the associated assets can be instantly monitored using Datapoints, which correspond to possibly time-varying properties of KIS.Devices. They can be also defined as exchanged variables between a KIS.Device and KIS.MANAGER.
- Performance of the system and the associated assets can be periodically determined using Datapoint-based key performance indicators (KPIs).
- It allows defining users as human beings with granted access determined by membership to a given user group.
- System behaviour and performance can be visualized using a set of time-driven plots and aggregated charts.
- E-mail notifications pertaining to system behaviour can be predefined and automatically distributed.
- A high security level is attained with Message Queuing Telemetry Transport (MQTT), which is a standard messaging protocol for the IoT. Moreover, designated certificates are used for secure authentication.

Having a general overview of both hardware and software functionalities, let us proceed to onboarding, i.e., the process of linking KIS.Devices with the KIS.MANAGER. However, a preliminary step towards onboarding is to register at the KIS.MANAGER, which can be easily realized with <https://kismanager.rafi.de>. Once a user company account is created, a compulsory checklist should be verified:

- a KIS.Device,
- an M12-to-USB cable (see Table 2.4),
- a computer/tablet equipped with a web browser and a USB port,
- a company account with admin user rights (see Sect. 2.3),
- WLAN access with permission credentials.

After completing the compulsory checklist, one can perform the onboarding procedure:

Step 1. Connection:

1. Plug-in KIS.Device to a PC/tablet using an M12-to-USB cable.
2. The KIS.Device status LED color should change from red into yellow (see Table 2.1).
3. The KIS.Device is available in the PC/tablet as a mass storage device.

Step 2. Authentication:

1. Open <https://kisme.rafi.de/en/> and go to the Onboarding link.
2. Enter your login and password pertaining to KIS.MANAGER admin rights.
3. Enter your designated WiFi parameters: SSID, password and WLAN encryption mode.
4. Click the save button to generate onboarding.zip containing WiFi parameters and store it onto the PC/tablet.

Step 3. Upload onboarding.zip onto the KIS.Device visible as a mass storage device.

Step 4. Processing onboarding.zip:

1. In progress: the status LED starts to flash in yellow;
2. Completed: the status LED lights constantly in yellow.

Step 5: Onboarding completion:

1. Under an available WiFi connection, the KIS.Device status LED should perform the following cycle: Yellow → Magenta → Blue → Green (see Table 2.1).
2. After refreshing the KIS.MANAGER, the new KIS.Device is available in Main menu → Assets.

> Changing WLAN

The WLAN data, and hence the WiFi network, can be changed on demand by simply repeating the above Step 1–Step 5 onboarding procedure.

> Accessing the KIS.ME demo

It is possible to preview KIS.ME performance without having your own KIS.Devices. For that purpose, a KIS.ME demo platform was designed, which can be accessed by performing the following steps:

1. Go to <https://kisme.rafi.de/en/#demo>.
 2. Access the KIS.ME demo with
 - Username: `demo.kisme@rafi.de`.
 - Password: `Demo1234!`.
-

2.2 Hierarchical Structure: From Assets and Users to Workspaces

The objective of this section is to provide a general KIS.ME-based system structure overview. For that purpose, let us introduce a suitable nomenclature:

User group: a set of users with a predefined KIS.MANAGER rights level,

Asset group: a set of assets,

Workspace: a selected part of the system, which inherits its desired set of assets.

From the above definitions, it is evident that Workspace and Asset group can be somehow perceived as synonyms. Indeed, while going to Main menu → Asset groups, one can see the view which is portrayed in Fig. 2.6. Thus, all system assets associated

Asset Group ▲	Definition
<u>My Devices</u>	Inventory
<u>Workspace 1</u>	Workspace
<u>Workspace 2</u>	Workspace
<u>Workspace 3</u>	Workspace
<u>Workspace 4</u>	Workspace
<u>Workspace 5</u>	Workspace

Fig. 2.6 Asset groups

with all KIS.Devices are contained in the inventory asset group My devices. KIS.ME, which makes it possible to arrange five workspaces (see the column Definition in Fig. 2.6) may inherit the assets contained in the inventory asset group.

> Adding asset groups

The current licence model allows six asset groups, i.e., My inventory, Workspace 1–Workspace 5. In the prospective licence models, it will be possible to add new groups.

Once Asset groups are defined, it is possible to proceed to User groups, which can be reached through Main menu → User groups. As can be seen in the column Description in Fig. 2.7, there are four predefined user groups: Admin, Installer, Operator and Observer, possessing various rights and permissions (see Sect. 2.3 for details). Thus, a single user may belong to these groups, which strictly defines rights and permissions. On the other hand, all five workspaces can also be perceived as user groups.

> User group membership

A single user may belong to multiple user groups, which define rights and permissions. This also means that he can belong to multiple user groups associated with workspaces, which clearly determine access to the desired asset groups.

Fig. 2.7 User groups

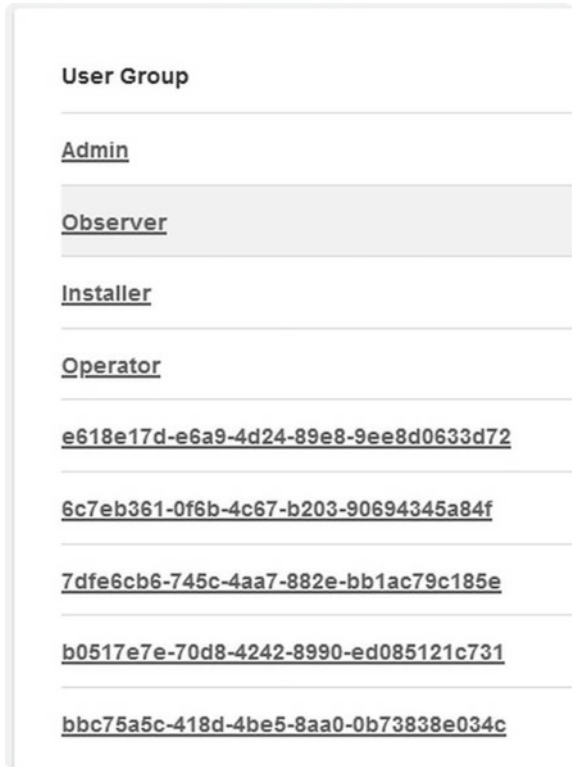
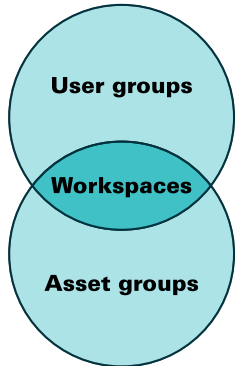


Fig. 2.8 KIS.ME hierarchical structure



Finally, the KIS.ME hierarchical structure can be expressed using Fig. 2.8. Thus, the objective of the subsequent section is to provide a concise overview pertaining to rights and permissions acting inside this structure.

2.3 Rights and Permissions

Let us start with four predefined user groups (see Fig. 2.7), which are initially called Admin, Installer, Operator and Observer. Their concise overview is given in Table 2.5, which uses the following nomenclature:


- Access: the user has access to a given feature;
- Add/delete: the user is able to add and/or deleted a given feature;
- Manage: the user is able to manage the properties of a given feature;
- View: the user is able to view a given feature.

➤ Names and permissions of predefined user groups

It should be noted that the initial names of predefined user groups can be modified. Alterations can be also performed with respect to their permissions (see Fig. 2.14).

2.3.1 User Management

The process of adding a user is very intuitive but requires access with admin rights (see Table 2.5). Under such a condition, adding a new users and assigning them the desired rights boils down to the following steps:

1. Go to Main menu → User management → Users.
2. Push the Create new user button. 
3. In the Master data tab (see Fig. 2.9), provide e-mail, language (locale), time zone, full name, and initial state.

➤ Initial state

The Initial state field determines whether the new user must agree (terms of acceptance pending) to an end user license agreement or whether this can be omitted and the user can be active immediately (Active).

4. In the User groups tab (see Fig. 2.10), push the Assign to User Groups button.
5. By selecting the desired checkboxes, assign a user to a predefined group (Admin, Installer, Operator, Observer; cf. Table 2.5) and to desired workspaces (Fig. 2.11).

Table 2.5 Rights of predefined user groups

Rights	Admin	Installer	Operator	Observer
Add/delete assets				
Manage permitted assets	x	x		
Access to all asset groups	x			x
Access to permitted asset groups	x	x	x	x
Add/delete users	x			
Add/delete dashboards	x	x		
Manage permitted dashboards	x	x	x	
View permitted dashboards	x	x	x	x
Add/delete digital twins	x	x		
Manage permitted digital twins	x	x	x	
Add/delete CDPs and/or KPIs	x	x		
View CDPs and/or KPIs	x	x	x	x
Add/delete e-mail templates	x			
Manage permitted e-mail templates	x			
Add/delete rules in rule engine	x	x		
Manage rules in rule engine	x	x		
View rules in rule engine	x	x	x	x

Fig. 2.9 Creating a new user: master data

The screenshot shows a 'Create User' form with two tabs: 'Master Data' (selected) and 'User Groups'. The form contains the following fields and options:

- E-mail ***: A text input field.
- Locale ***: A dropdown menu with 'en-US' selected.
- Time Zone ***: A dropdown menu with 'Europe/Berlin' selected.
- Full Name**: A text input field.
- Initial state**: Two radio buttons: 'Terms Acceptance Pending' (unselected) and 'Active' (selected).
- * required**: A label indicating that the fields marked with an asterisk are required.
- Buttons**: 'Save' (grey), 'Cancel' (white), and 'Toggle hidden properties' (text).

> Admin and workspaces

Irrespective of the workspaces being assigned, a user with admin rights has access to all of them (see Table 2.5).

6. After pushing the Save button, the user will receive an e-mail that contains a link which enables providing a password.

Finally, any modification pertaining to an existing user can be realized with the above procedure. However, instead of Step 2, an existing user has to be selected from the available user list.

Fig. 2.10 Creating a new user: user groups

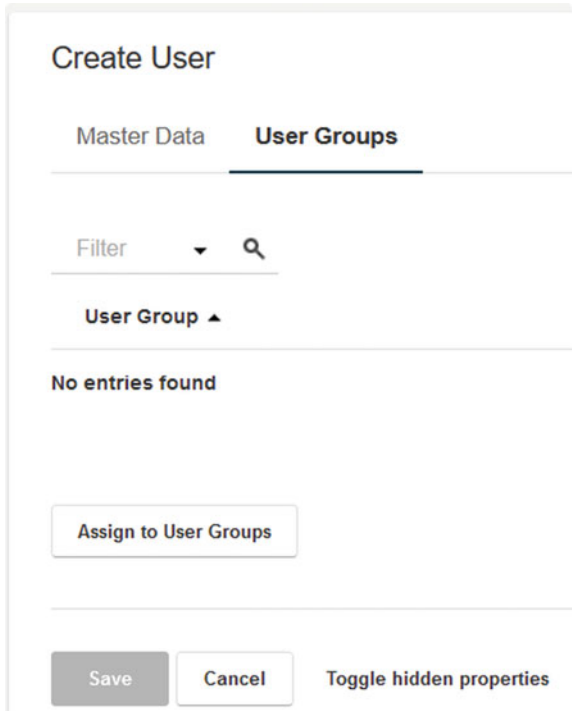


Fig. 2.11 Assigning a user to user groups

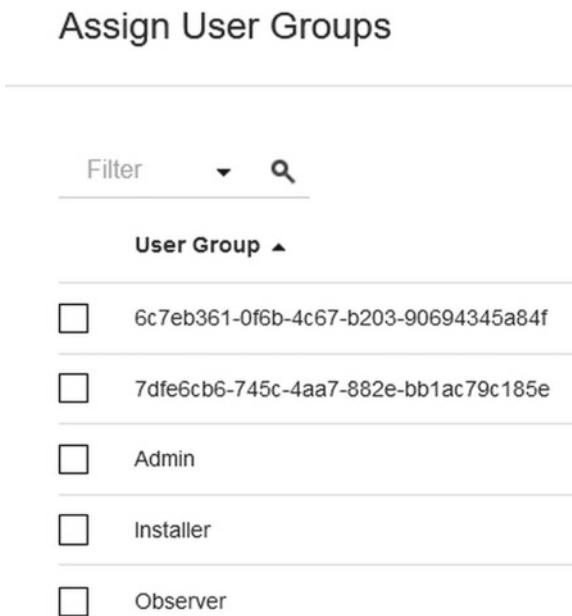
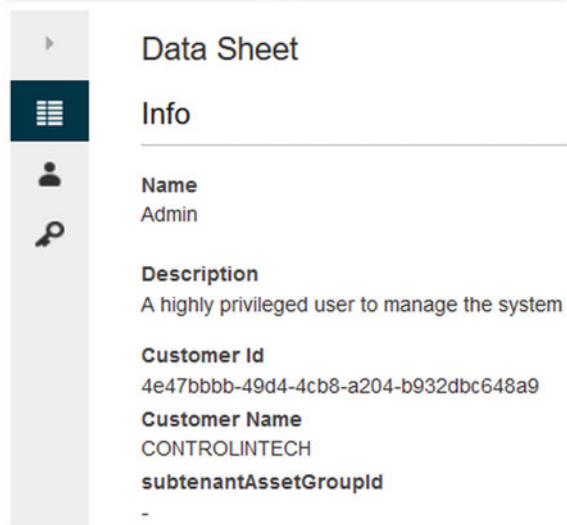



Fig. 2.12 User group data sheet



2.3.2 User Groups and Workspace Management

As mentioned in the preceding section, each user can belong to multiple user groups. The permissions of a given user group to another one can also be freely defined. Indeed, while going to Main menu → User management → User groups, one can see the view portrayed in Fig. 2.7. Subsequently, by selecting any group, e.g., admin, one can see the view shown in Fig. 2.12, which contains essential details about this group. By selecting the Edit button , one can see the view presented in Fig. 2.13. As can be observed, there are two tabs:

- Info: it contains the name, description and other descriptive parameters of a group;
- User Groups with Access Permissions: it pertains to the list of user groups for which a given access group has access permission (see Fig. 2.14).

> Assigning users to a group


It should be mentioned that by proceeding to Main menu → User management → User Groups and then selecting a desired group, one can see the view portrayed in Fig. 2.12. By pushing the Assigned Users button , it is also possible to edit the user assignment.

Fig. 2.13 Editing a user group: info

Edit User Group

Info User Groups with Access Permission

User Group

Description

subtenantAssetGroupId

Not changeable once set and saved

Customer Id


Not changeable once set and saved

Customer Name

Not changeable once set and saved

Advanced settings

Technical name (Autogenerated) *

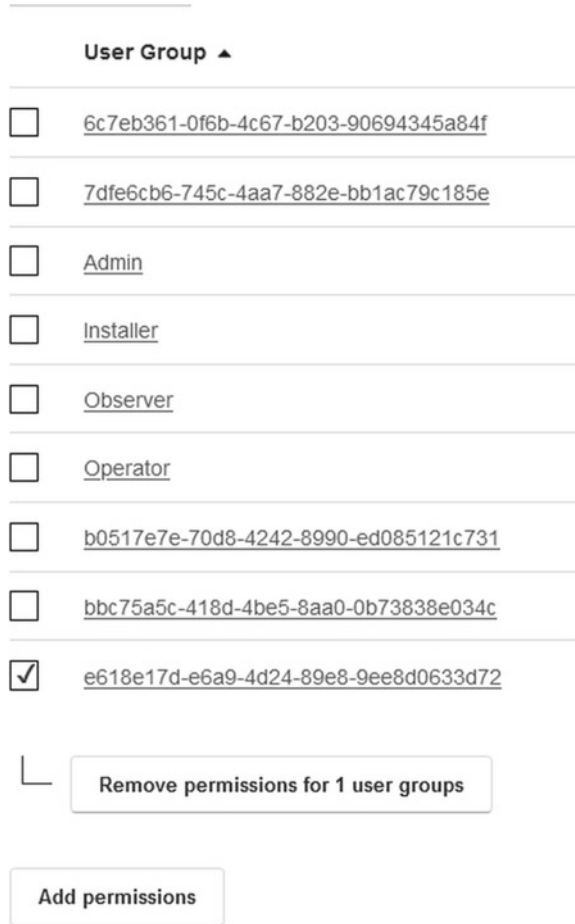
 
This field is tenant wide unique

*** required**

2.4 Asset Management



The objective of the preceding sections was to perform a suitable introduction to the hierarchical KIS.ME structure along with the KIS.Device onboarding procedure. As a result of performing onboarding on a set of assets, one can see a view similar to that presented in Fig. 2.15. Indeed, it can be easily accessed with Main menu → Assets. This sample view indicates that there are eight KIS.Devices, i.e., four KIS.BOXes and

Fig. 2.14 Editing a user group: user groups with access permissions



four KIS.LIGHTs. It can be also immediately deduced (see the Connection column) that, except for KIS.BOX 0 and KIS.LIGHT 3, all KIS.Devices are connected with KIS.MANAGER via dedicated WiFi(s). Subsequently, by selecting a sample asset, e.g., KIS.BOX 1, one can see the view presented in Fig. 2.16. The objective of the subsequent part of this section is to perform essential asset management concerning

- changing the name of the asset,
- assigning the asset to asset groups,
- obtaining information about current status of an asset.

Let us start with the first task by pushing Data Sheet button . As a result, the view presented in Fig. 2.17 is obtained. Subsequently, the Master Data edit button  can be used to change the name of the asset as shown in Fig. 2.18. Let us proceed to the second task, which pertains to assigning KIS.BOX 1 to desired asset groups. For that purpose, the Asset groups tab should be selected as shown in Fig. 2.19. Finally,

Asset Name ▲	Connection	Asset Groups
KIS.BOX 0	Offline	My Devices
KIS.BOX 1	Online	My Devices
KIS.BOX 2	Online	My Devices
KIS.BOX 3	Online	My Devices
KIS.LIGHT 0	Online	My Devices
KIS.LIGHT 1	Online	My Devices
KIS.LIGHT 2	Online	My Devices
KIS.LIGHT 3	Offline	My Devices

Fig. 2.15 Asset view

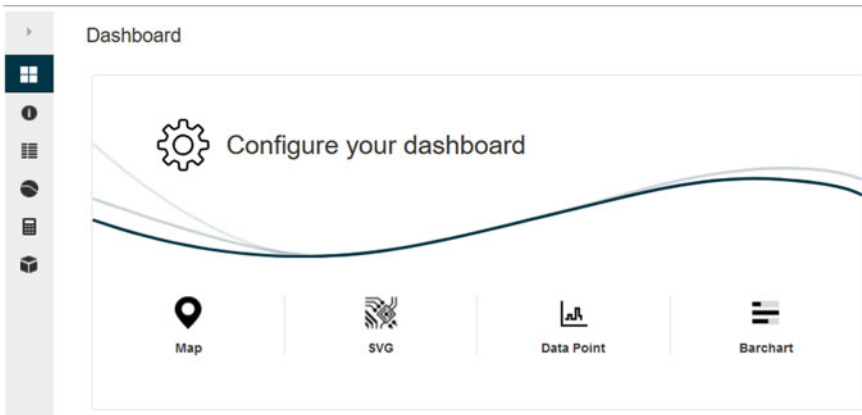


Fig. 2.16 Asset view: KIS.BOX 1

the assignment process reduces to pushing the Assign to Asset Groups button and selecting the desired asset groups as depicted in Fig. 2.20. The process of unassigning an asset from the asset group can be performed in a similar fashion. Indeed, it is enough to check a desired checkbox and push the Unassign button.

Fig. 2.17 Data sheet of KIS.BOX 1

The screenshot displays a mobile application interface for viewing asset data. On the left is a vertical sidebar with icons for navigation: a right-pointing arrow, a grid, an information icon, a list icon (highlighted in dark blue), a pie chart, a calculator, and a cube. The main content area is titled 'Data Sheet' and 'Master Data'. It shows the following information:

- Asset Name:** KIS.BOX 1
- URN:** urn:rafi:sbox:9c65f93cc3eb
- Show QR Code:** A button to view the QR code.
- Device Type:** KIS.BOX
- Asset Groups:** A section with a filter dropdown and a search icon, showing 1 group.
- Asset Group:** A dropdown menu currently showing 'My Devices'.
- Asset Shadow Datapoints:** A section with the following values:
 - isOnline:** true
 - lastUpdate:** 06/20/2021 14:00:08 (+02:00)
 - appVersion:** r32
- Asset Shadow Values:** A section with the following value:
 - connection.last:** 06/20/2021 14:00:05 (+02:00)

Fig. 2.18 Changing the asset name

Edit Gateway

Data Sheet Asset Groups Onboarding

Asset Name *

URN *

Transport channel *


Use certificate onboarding

Device Type *

Not changeable once set and saved

*** required**

> Group relationship graph

The relationship between an asset and the associated asset groups can be easily visualized. Indeed, in Fig. 2.17, one can find the Group relationship graph button , which can be used to visualize an associated graph. Such a sample graph is portrayed in Fig. 2.21.


The last task pertains to obtaining information about detailed parameters of an asset by pushing the Info button  (see Fig. 2.16). The above parameters cover the device information divided into the following groups:

Fig. 2.19 Asset and the associated asset groups

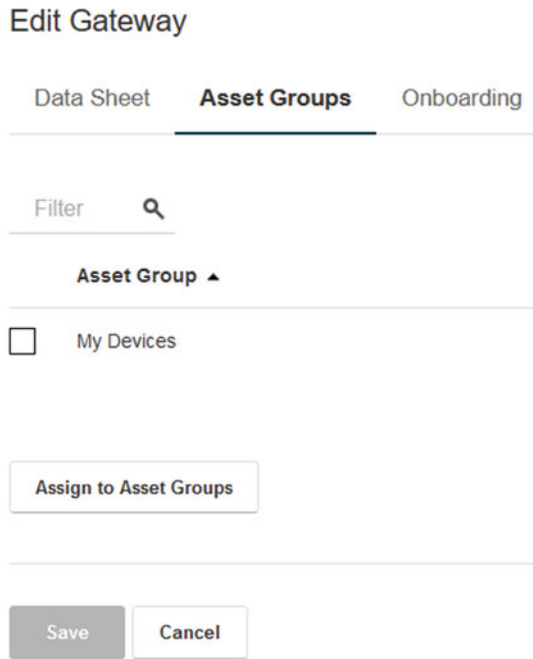


Fig. 2.20 Assigning asset groups

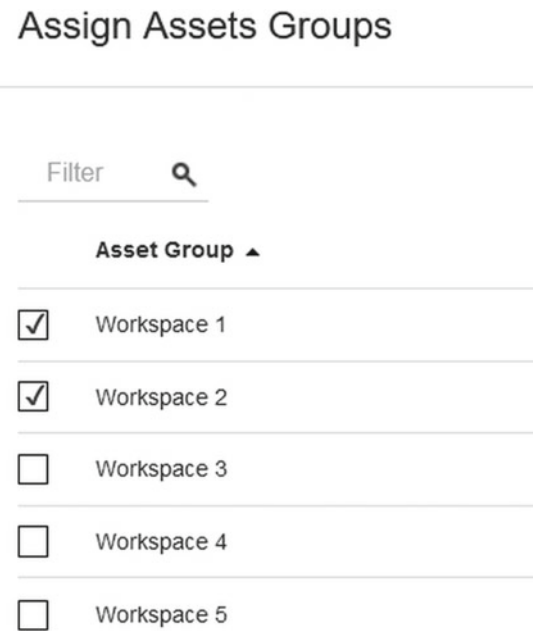
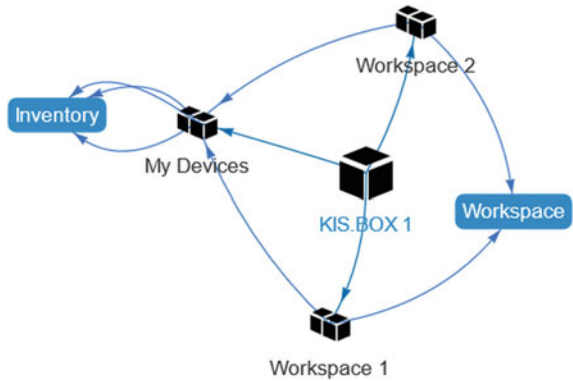



Fig. 2.21 KIS.BOX 1: group relationship graph



Hardware: type (KIS.BOX/KIS.LIGHT), part number, serial number, data matrix code, MAC address, hardware revision,
Software: OS version, application version, microcontroller firmware version,
Network: WiFi SSID, WiFi signal strength, WiFi channel, IP address, subnet, gateway,
Firmware update: a set of detailed parameters including the update status,
Certificate: the certificate expiration date of a KIS.Device.

2.5 Dashboards and Widgets

The view presented in Fig. 2.16 is divided in the so-called Dashboards, which are defined as an overview pages for an asset and/or asset groups. Dashboards can be managed by pushing the Edit dashboard button . After selecting this option, one can perform one of the following tasks (see Fig. 2.22):

- add a new dashboard,
- design or edit an existing dashboard.

The first one is very intuitive and does not need any further explanation as it reduces to providing the name of a new dashboard. As a result, the dashboard is automatically created and displayed as a new tab in the asset view. Thus, let us proceed to designing a dashboard. Each one is composed of widgets. A widget is a component of the interface which makes it possible to perform a desired action. There are nine available widget types, which can be characterized as follows (see Fig. 2.23):

- Digital twin,
- Info,
- Datapoint Chart,
- Data Sheet,

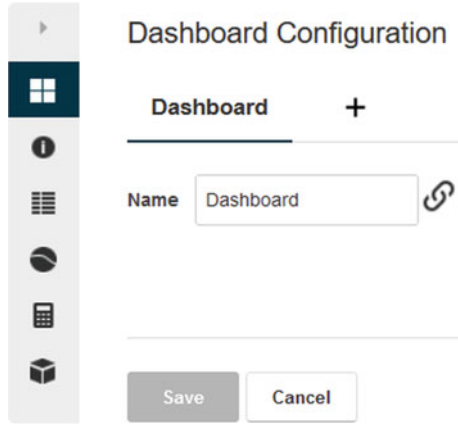


Fig. 2.22 Dashboard design

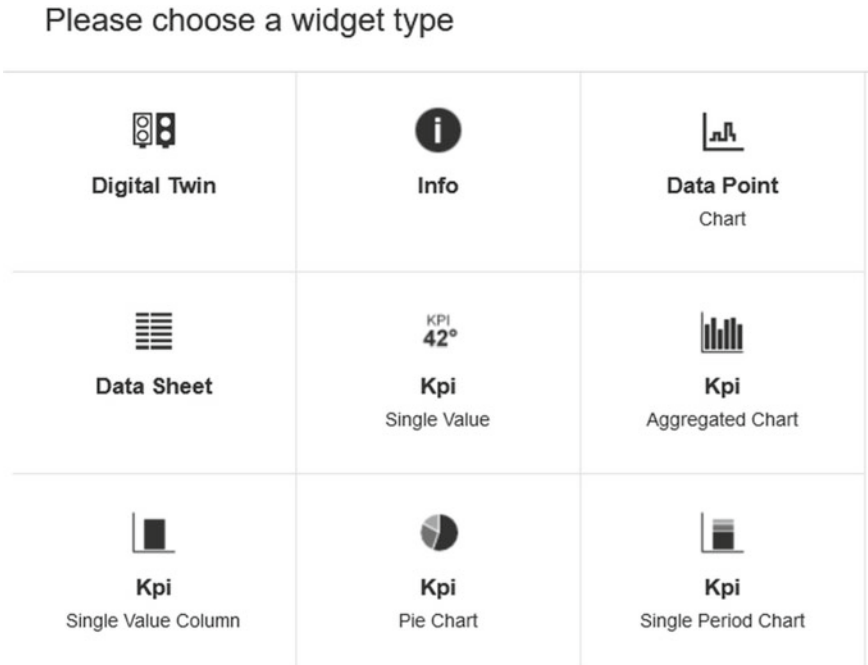



Fig. 2.23 Asset: nine widgets

- KPI Single Value,
- KPI Aggregated Chart,
- KPI Single Value Column,
- KPI Pie Chart,
- KPI Single Period Chart.

The first four widgets can be directly used for digitalization of KIS.Devices as well as the monitoring and analysis of their behaviour. The remaining five widgets require suitable preprocessing using KPIs, which are discussed in Sect. 4.1.2.

2.6 Digital Twin Design

The objective of this section is to introduce the Digital twin widget along with its essential functionalities. The digital twin can be defined as a KIS.MANAGER-based virtual counterpart of a KIS.Device, which is connected to the real one through dedicated WiFi. A graphical representation of both KIS.Device digital twins is presented in Fig. 2.24. Now, let us proceed to digital twin design. For that purpose it is necessary to go to Main menu → Assets and select the desired asset, e.g., KIS.BOX 1. Subsequently, the Edit dashboard button  should be used and then the Add widget button can be employed, resulting in the view portrayed in Fig. 2.23. Finally, the digital twin design boils down to selecting an appropriate widget and applying the resulting dashboard changes. This produces in the dashboard view presented in Fig. 2.25. The same procedure can be performed for any KIS.LIGHT, e.g., KIS.LIGHT 1. As a result, the digital twin presented in Fig. 2.26 is obtained. Irrespective of the KIS.Device being used, it can be noticed that the actual values of both digital inputs (GPIO 3, GPIO 4) and outputs (GPIO 1 and GPIO 2) are given as well. In particular, a 0 binary state is signified by Off while 1 is denoted by On. Moreover, their switching frequency, expressed in mHz, can be observed as well. As detailed in Sect. 2.1, the possible operational LED colors are limited to the ones provided in Table 2.2. As shown in Figs. 2.27–2.28, the digital twins allow changing the color of the operational LEDs by simply selecting the desired one and then pressing the Set button. Note also that a given operational LED may be flashing or blinking, which can be achieved via the Flashing checkbox. It should be also noted that the digital twins display the current state of the operational LEDs, which can evolve in various ways, e.g., due to the appropriate rules implemented within Rule engine (see Sect. 2.9).

Time drive

Let us perform a simple change of the KIS.BOX 1 state pertaining to its second button operational LED color (cf. Button 2 in Fig. 2.28). It can be realized as follows:

1. Select the Button 2 color as blue.
2. Press the Set button.

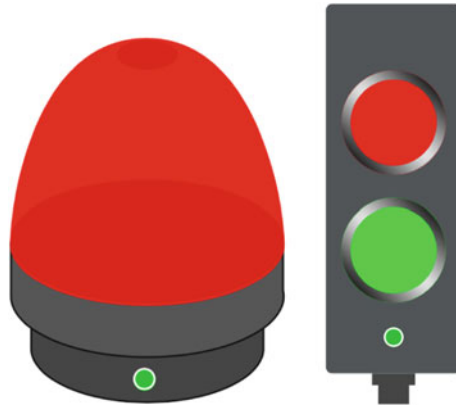


Fig. 2.24 KIS.LIGHT and KIS.BOX digital twins

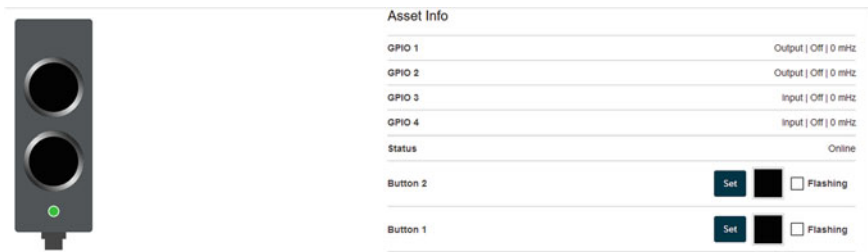


Fig. 2.25 Digital twin of KIS.BOX 1



Fig. 2.26 Digital twin of KIS.LIGHT 1

- 3. Wait a moment.
- 4. Select the Button 2 color as black.

After this simple procedure, one can press the Start time drive button . This allows monitoring or reconstructing historical states of KIS.Devices, which can be realized according to Fig. 2.29.

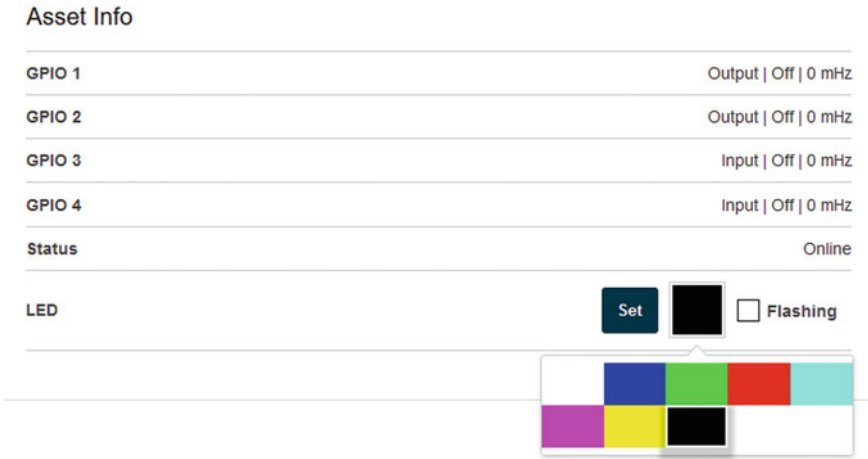


Fig. 2.27 Changing the KIS.LIGHT operational LED color

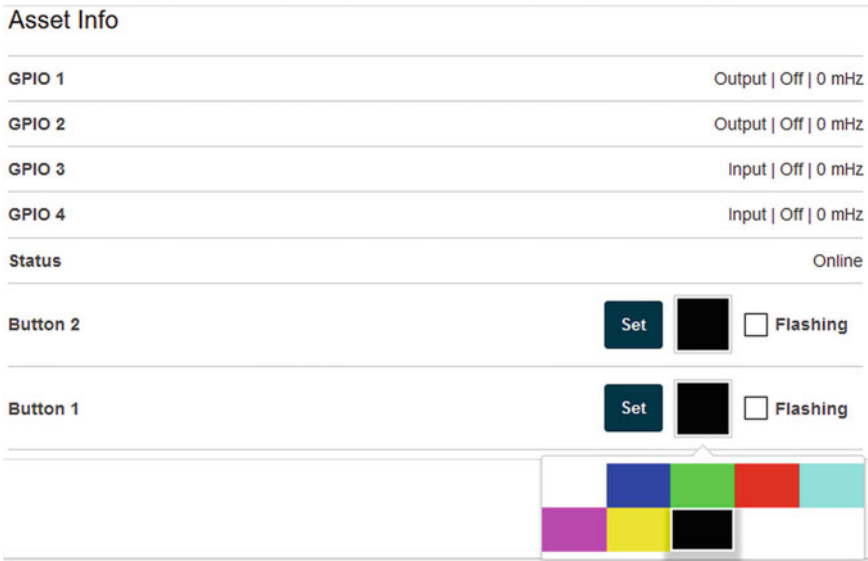


Fig. 2.28 Changing the KIS.BOX operational LED color

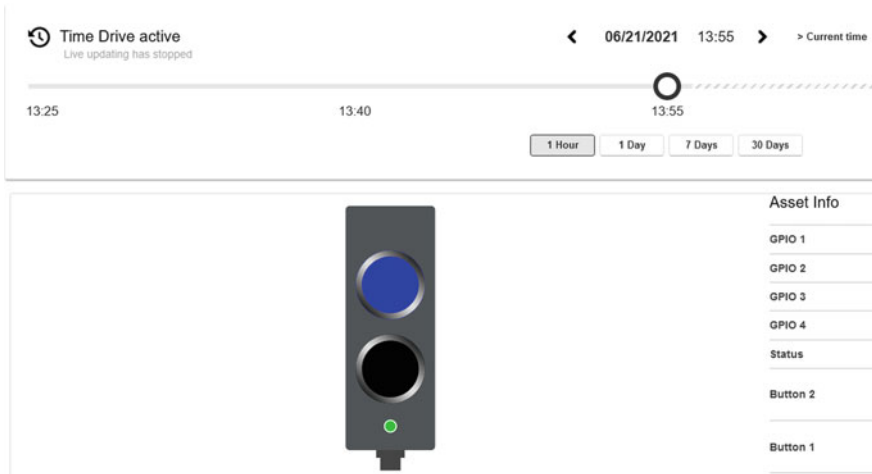


Fig. 2.29 KIS.BOX 1 time drive

The objective of the preceding sections was to introduce the reader into essential subjects related to asset and user management. The subsequent part aims at going into details pertaining to the description of the current asset state using the concept of Datapoints.

2.7 Datapoints: Plotting and Storing Data

Datapoints can be perceived as links between KIS.Devices and KIS.MANAGER. They can be easily accessed through Main Menu → Assets → KIS.Device → Datapoints, and can be of different types, which are listed in Table 2.6.




Table 2.6 Datapoint types

Type	Description
Boolean	A logical value, i.e., either <code>true</code> or <code>false</code>
Long	An integer value, e.g., 24
Double	A double precision floating point value, e.g., 3.14
Text	A character string, e.g., #0000FF

> Important

From the software engineering viewpoint, Datapoints can be perceived as read only variables, which can be further processed and analysed. The only restrictions are the following:

- Neither logical nor numerical operations on the Text type Datapoints are allowed, and hence they can only be stored or visualized.
- No numerical operations can be performed on the Boolean type Datapoints; however, this limitation can be easily tackled with the `IF []` command (see A.16).

Datapoints are processed in real time (limited by the data transfer rate), and hence their values depend on the current state of a KIS.Device. A full list of Datapoints along with their simple sample applications is provided in Appendix B. The evolution of Datapoints can be easily observed by going to Main menu → Assets → KIS.Device and then pressing the Datapoints button . As a result, the view presented in Fig. 2.30 is obtained. Subsequently, by selecting the desired Datapoints, their time evolution can be graphically observed in a dedicated plot. This process is illustrated in Fig. 2.31. A similar functionality can be directly obtained within the KIS.Device dashboard. Indeed, by proceeding to the dashboard, i.e., by pressing the Dashboard button  and then the Edit dashboard one , it is possible to add one of the widgets (Add widget button) presented in Fig. 2.23. Finally, to achieve the desired functionality, a Datapoint Chart is incorporated within the dashboard. Its configuration requires selecting a Datapoint (see Fig. 2.32), providing a headline of the figure as well as the plotting interval. After this preliminary setup, one can proceed to defining the plot options, which can be realized according to Fig. 2.33. Apart from these, one can set the plot color as well as define the axis properties (cf. Fig. 2.34):

Show axis: enable/disable an axis;

Scale axis: an axis may have a limited range. That can be time-varying, which can be realized by assigning a suitable Datapoint;

Show Min/Max: show minimum and maximum values of the data being plotted.

Apart from the above features, it is possible to define Thresholds over/below which the plot color will be changed (see Fig. 2.34).

> Multiple plots

The datapoint Chart widget allows presenting multiple plots, which can be individually managed using options and properties described in this section.

Data Points

Name ▲	Type
<input type="text" value="Filter..."/>	- All - ▼
 appVersion	 Datapoint
 bootloaderVersion	 Datapoint
 button1Color	 Datapoint

Fig. 2.30 KIS.BOX 1 datapoints

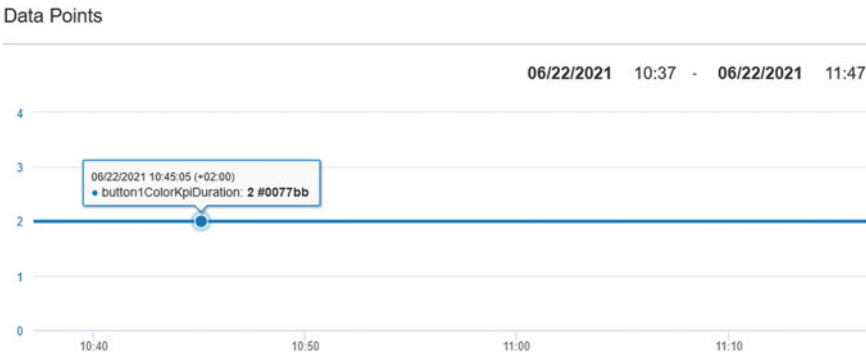







Fig. 2.31 KIS.BOX 1 datapoints' trend

Fig. 2.32 Datapoint chart: selecting a datapoint

Please select a Data point...

Name ▲	Type
<input type="text" value="Filter..."/>	- All - ▼
appVersion	 Datapoint
bootloaderVersion	 Datapoint
button1Color	 Datapoint
button1ColorKpi	 Datapoint
button1ColorKpiDuration	 Datapoint

Plot style	stairs	linear	shaded stairs	shaded linear
Point style	line	circle	square	triangle
Line style	solid	dotted	dash-dot	dashed

Fig. 2.33 Datapoint chart: plot options and their interpretation

button1ColorKpiDuration

Display

Show Axis

Scale Axis

Show Min/Max

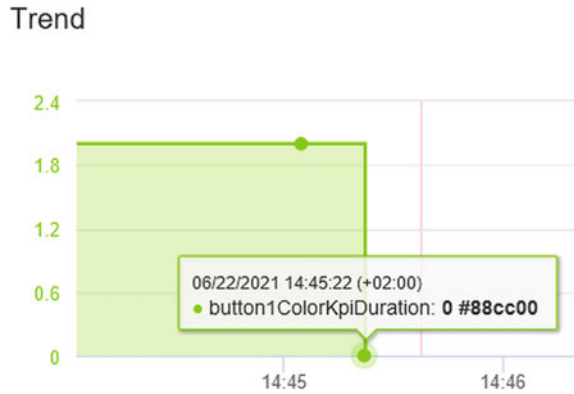
Start

Threshold 1 e.g. 70

Threshold 2 e.g. 70

Fig. 2.34 Datapoint chart: axis and colors

Fig. 2.35 Datapoint chart:
an example



Practical example

The illustrative example being considered aims at realizing the following steps:

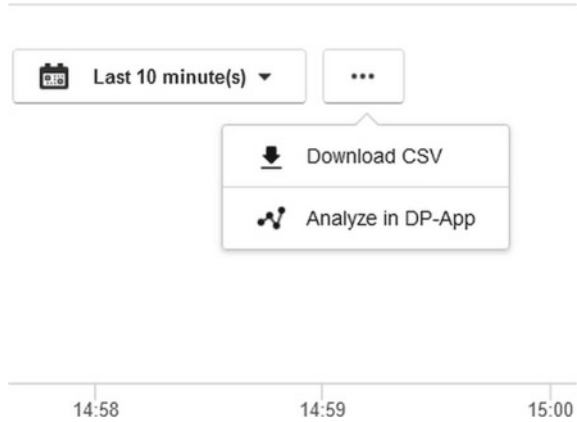
1. Go to the KIS.BOX 1 dashboard.
2. Add a new Datapoint Chart widget.
3. Select the `button1ColorKpiDuration` Datapoint.
4. Set the following plot options:
 - Plot style: shaded stairs,
 - Point style: circle,
 - Lines style: solid.
5. Set the plotting interval to 10 min.
6. Save the dashboard with the new widget.
7. Observe the current value of the Datapoint and verify it with Table 2.2.
8. Wait a moment, change the KIS.BOX 1 operational LED color to blue, verify the current value of the Datapoint and compare it with Table 2.2.

The obtained results are shown in Fig. 2.35. Let us proceed to check the obtained results with Table 2.2. Initially, the KIS.BOX Button 1 operational LED color was black, and hence one can see the line at the level of 2. Similarly, after a moment of time, the color was set to blue, which corresponds to the 0-valued point.

➤ Storing and analysing data

As shown in Fig. 2.36, additional features of the Datapoint Chart are as follows :

Fig. 2.36 Datapoint chart: storing and analysing data




Download CSV: the data can be stored as a CVS file, with the first column being a time stamp and the remaining columns corresponding to the values of the associated Datapoints;

Analyze in DP-App: this feature moves the user to the Datapoint view like the one presented in Fig. 2.31.

2.8 Let Us Go to Workspaces: An Introductory Example with the Floorplan Widget


The objective of this section is to introduce a crucial feature of asset groups concerning the possibility of visualizing the system floorplan. It can be simply defined as a graphical representation of the workshop. The floorplan is virtually implemented within KIS.MANAGER using the Floorplan widget. To access it, it is necessary to go to Main menu → Asset Groups, which results in the view presented in Fig. 2.37. Subsequently, the desired asset group has to be selected, e.g., Workspace 1.

> Floorplan vs. assets


The floorplan can only contain the assets which are assigned to a given group. For a comprehensive description pertaining to asset management, the reader is referred to Sect. 2.4. Alternatively, it is possible to perform this task directly from a workspace by simply pushing the Assigned Assets button . Subsequently, the following steps should be realized:

Asset Group ▲	Definition
<u>My Devices</u>	Inventory
<u>Workspace 1</u>	Workspace
<u>Workspace 2</u>	Workspace
<u>Workspace 3</u>	Workspace
<u>Workspace 4</u>	Workspace
<u>Workspace 5</u>	Workspace

Fig. 2.37 Asset groups

1. Push the Edit assigned assets button .
2. Push the Add assets to this group button.
3. Select the desired assets and add them to the group.

Note that the process of removing assets from a group can be realized in an analogous way.

After selecting the workspace, e.g., Workspace 1, one can see the view presented in Fig. 2.38. Similarly as in Sect. 2.5, dashboards can be managed by pushing the Edit Dashboard button . After selecting this option, it is necessary to push the Add widget button, which results in the view presented in Fig. 2.39. However, the resulting set of widgets is different than the one described in Sect. 2.5 (see Fig. 2.23). Indeed, there are the following nine widgets:

- Floorplan,
- Datapoint Chart,
- Data Sheet,
- Aggregation,
- KPI Single Value,
- KPI Aggregated Chart,
- KPI Single Value Column,
- KPI Pie Chart,
- KPI Single Period Chart.

The first three widgets can be directly used for digitalization of asset groups as well as the monitoring and analysis of their behaviour. The remaining five require suitable preprocessing using KPIs, which are discussed in Sect. 4.1.2.

Let us start by selecting the Floorplan widget. This requires an appropriate graphical representation of the real floorplan in the form of an SVG file. Such a kind of

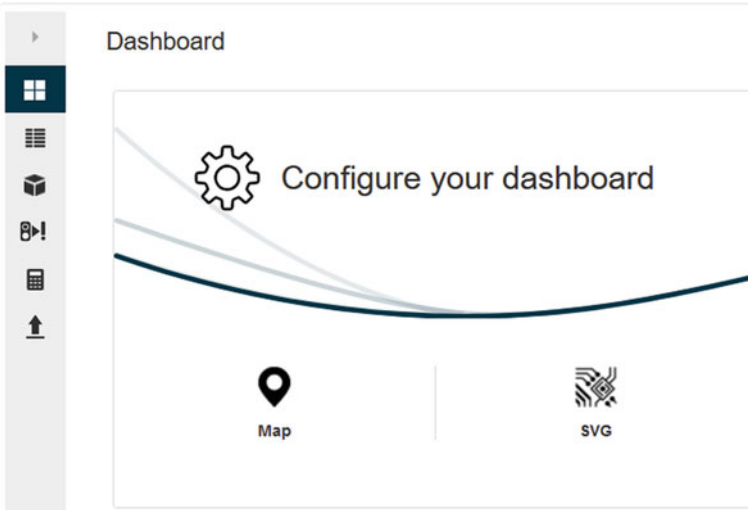


Fig. 2.38 Asset groups: workspace 1

Please choose a widget type










 Floorplan A group floorplan	 Data Point Chart	 Data Sheet
 Aggregation	 Kpi Single Value	 Kpi Aggregated Chart
 Kpi Single Value Column	 Kpi Pie Chart	 Kpi Single Period Chart

Fig. 2.39 Workspace: nine widgets

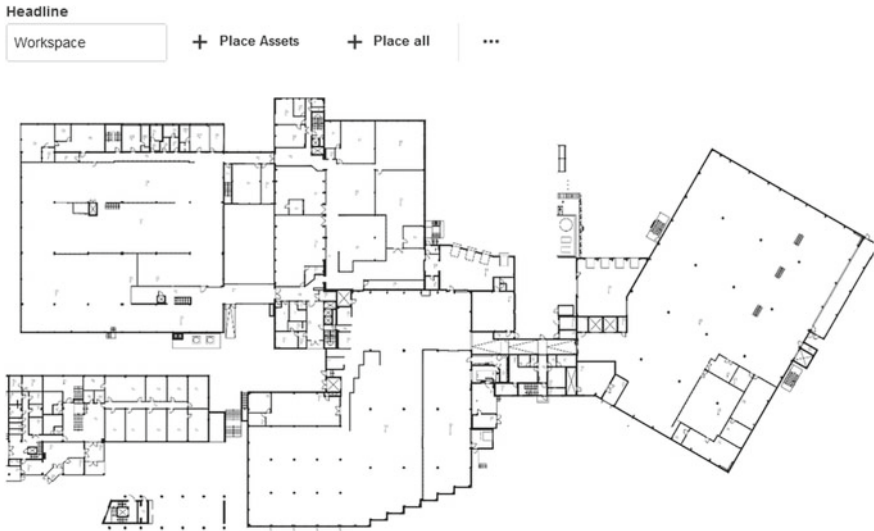



Fig. 2.40 Floorplan widget: an initial configuration

files employs a two-dimensional vector graphic format created by the World Wide Web Consortium. It expresses images with a text format that is based on XML. There are plenty of free and commercial tools which can be used for preparing a floormap using the SVG format. A good representative example is the freely available Inkscape package [1]. Having an SVG-based floorplan, it is possible to use the Floorplan widget. Its initial configuration reduces to providing a desired Headline and the above-mentioned SVG image. As a result, the view portrayed in Fig. 2.40 is obtained. Subsequently, either all or selected assets can be introduced within the floorplan. Let us proceed with selected assets. To perform this action, it is necessary to use the Add widget button (cf. Fig. 2.40). The desired assets can be added as depicted in Fig. 2.41. Finally, the assets (KIS.BOX 1 and KIS.LIGHT 0) can be freely located within the floorplan, which results in the dashboard presented in Fig. 2.42.

> Asset group time drive

Similarly as in Sect. 2.6, it is possible to monitor or reconstruct historical states of the asset group. This can be easily realized by pressing the Start time drive button  (cf. Fig. 2.38).

Change Settings

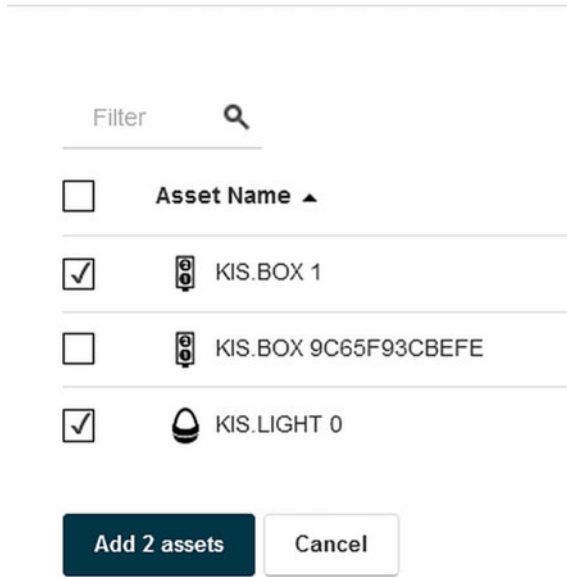


Fig. 2.41 Floorplan widget: adding assets

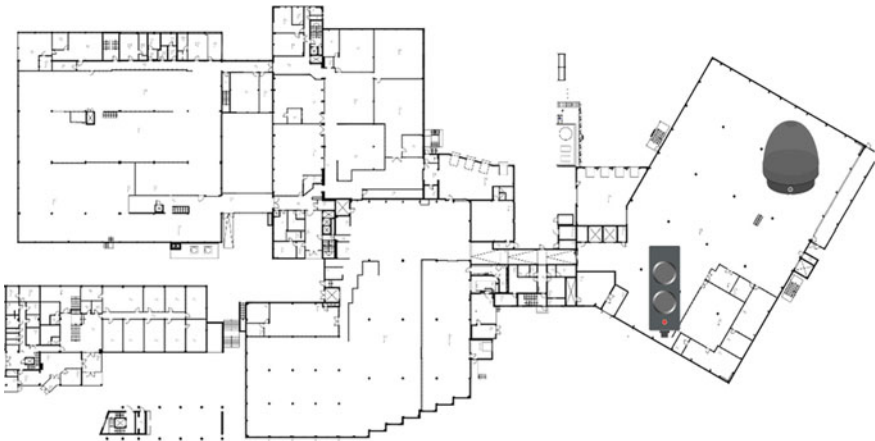


Fig. 2.42 Floorplan widget within the dashboard

2.9 Let Us Rule: Managing Rules Within a Workspace

A rule-based system can be perceived as a way of transforming a human expert's knowledge into an automated framework [2, 3]. For the purpose of KIS.ME, such a framework is called Rule engine. On the other hand, it can be seen as a KIS.MANAGER functionality, which makes it possible to implement IF-THEN rules governing interactions between assets. Thus, the rule-based system can be simply designed using a set of assets and a set of rules dictating their behaviour. In KIS.ME, rules are exemplified as a set of IF-THEN statements labelled with a unique name:

$$\text{rule name: IF antecedent THEN consequent.} \quad (2.1)$$

Generally, a rule may have multiple antecedents linked by or and/or or operators. Similarly, it may have multiple consequences. For example,

$$\begin{aligned} \text{rule 1: IF A is black and C is white or X is green THEN} \\ \text{A is white, B is black.} \end{aligned} \quad (2.2)$$

Note that the antecedent of a rule possesses two parts:

1. a linguistic object (LO),
2. the value of the linguistic object.

The linguistic object is linked with its value through various operators, e.g., is, or mathematical operators: \leq , $<$, etc. Since KIS.ME operates on KIS.Devices, LOs can be designed either with KIS.BOXes or KIS.LIGHTs, which yields the following:

KIS.Device | Status | Operating Mode: it expresses functional access of a KIS.Device to WiFi and it can be either Online or Offline;

KIS.Device | GPIO | GPIO No: it corresponds to the logical status of a selected GPIO, either On (High) or Off;

KIS.BOX | Button No | Button No Color: it expresses the color of the button operational LED, which can take a value from Table 2.2 with an additional Boolean-valued Flashing option;

KIS.LIGHT | LED | LED Color: it expresses the color of the operational LED, which can take a value from Table 2.2 with an additional Boolean-valued Flashing option.

There are two operators linking LOs with their values:

EQUAL: checks if an LO has a given value;

NOT: checks if an LO does not have a given value.

Subsequently, a consequent can be defined as an Action, which can assign a value to one of the above-listed LOs except for the first one, i.e., KIS.Device | Status | Operating Mode. Another restriction is that only digital outputs can be set, i.e., KIS.Device | GPIO | GPIO 1 and KIS.Device | GPIO | GPIO 2. The above actions

should be perceived as the ones acting on a device. An action can also be associated with sending a predefined notification e-mail.

> Notification templates

A predefined notification e-mail is based on a notification template. Such a template can be defined by a user with admin rights (cf. Table 2.5) by simply going to Main menu → Portal Admin → Notification Templates. As a result, by using the Create new notification template button **+**, a notification template editor is obtained, which is presented in Fig. 2.43. The crucial features of such a template are as follows:

- Name: uniquely identifies a template within Rule engine actions;
- Subject: stands for the title of a predefined e-mail;
- Message: constitutes the body text of the predefined e-mail.

Both Subject and Message can be conveniently designed using a set of variables, which can be accessed after pushing the Add variables... button. The meaning of the crucial parameters should be interpreted as follows:

- asset.name: the name of a KIS.Device,
- asset.properties.type: either sBox or sLight,
- event.key: EMAIL_ACTION,
- event.timestamp?datetime: the date and time of an event.

Before proceeding to designing a sample rule, a set of suitable definitions has to be provided:

- Conditions: the set of antecedents merged with and/or operators,
- Triggers: the set of antecedents merged with or operators,
- Actions: the set of consequents.

Under the above definitions, triggers can be perceived as necessary conditions for performing a Rule engine-based inference. Additionally, triggers can be formed with all of the above-defined logistic objects. However, they can also use a linguistic object associated with pressing the KIS.BOX button. Unlike conditions, triggers verify if the value of a linguistic object has given instants, e.g., a button is pressed. Thus, a full list of triggers for linguistic objects is formed by extending the above-defined one with what follows:

- KIS.BOX | Button No | Pressed: it expresses the fact of pressing the KIS.BOX button.

Apart from the above functionalities, triggers have also optional settings:

- after x times: the trigger is fired when a given value of a linguistic object has been counted x times;

← Create Template

Name *

Type * E-mail

Languages

▾ default

Subject *

+ Add variables...

Message *

Fig. 2.43 Notification templates

Trigger ⓘ

Optional Settings

ⓘ

[Trigger Details ⓘ](#)

Fig. 2.44 Sample trigger with optional settings

after x minutes: the trigger is fired after x minutes from the time when a given value of a linguistic object has been recorded;
after x hours: the trigger is fired after x hours from the time when a given value of a linguistic object has been recorded.

Figure 2.44 shows a sample trigger, which is fired after pressing the KIS.BOX button two times. It should be also pointed out that this kind of trigger has an internal counter, which is automatically reset after reaching a given threshold. It can be also reset manually after using the Trigger details link along with the Reset button (cf. Fig. 2.45). Finally, let us note that such a manual reset is not available for the after x minutes and after x hours optional settings.



Fig. 2.45 Trigger reset

Sample rule

The objective of this example is to define a rule which satisfies the following requirements:

Environment: It is defined with Workshop 1 as well as employs KIS.BOX 1 and KIS.LIGHT 0.

Triggers: The triggers are associated with pressing KIS.BOX 1 Button 1 or Button 2.

Conditions: The operational LED color of KIS.LIGHT 0 can be either black or green and its status should be online.

Actions: The associated actions are as follows:

- change the operational LED color of KIS.LIGHT 0 to green;
- send a notification email to `john.doe@controlintech.pl` with the subject and title “Color change” while the body of the message being KIS.Device color has changed.

Let us start with defining an email notification template by going to Main menu → Portal Admins → Notification Templates. As has already been discussed, such a template can be designed according to Fig. 2.46. Before proceeding to Rule engine definitions, it can be observed that the above conditions may have a visible effect if the KIS.LIGHT 0 operational LED color is either black or green. Thus, an appropriate initial condition has to be imposed by going to Main menu → Assets, selecting KIS.LIGHT 0 and using its digital twin to set an appropriate operational LED color (Sect. 2.6). Under the above preliminary setup, triggers, conditions and actions can be intuitively defined by pushing the Rule engine button (cf. Fig. 2.42). Subsequently, the Create rule button should be used to open the Rule engine editor and provide the required ingredients, i.e., the name of the rule, triggers, conditions and actions. As a result, the view presented in Fig. 2.47 is obtained. After saving the rule, it is activated and operates within KIS.ME.

Create Template

Name * Type * E-mail

Languages

▾ default

Subject *

Message *

Fig. 2.46 Notification template: “change color”

> Rule interactions



As can be expected, each asset group/workspace has its own set of rules. However, when sharing assets between workspaces, the users must be cautious about their possible unappealing interactions.

2.10 State-Space Modelling

The objective of this section is to introduce the concept of the system state, which is a set of variables that can be used to describe any past and future system behaviour. Consequently, the system state-space is a space of admissible state values. Subsequently, the state-space model is defined as a set of rules which enables cyclical transition between the consecutive states.




Name: Exemplary rule Active:

Trigger

	KIS.BOX 1	Button 1	Pressed	×
	KIS.BOX 1	Button 2	Pressed	×

OR
×
+

Conditions

	KIS.LIGHT 0	LED	LED color	EQUAL	<input checked="" type="checkbox"/> Flashing	×
	KIS.LIGHT 0	LED	LED color	EQUAL	<input type="checkbox"/> Flashing	×
	KIS.LIGHT 0	Status	Operating Mode	EQUAL	Online	×

OR
×
+

Actions



	KIS.BOX 1	Set LED	Button 1 Color	<input checked="" type="checkbox"/> Flashing	×
	john.doe@controlintech.pl	Change color			×

Fig. 2.47 Sample rule

Traffic lights state-space model

To illustrate the concept of the state-space model, let us employ a traffic lights example. In this case, the transition rules can be clearly visualized using Fig. 2.48. The objective of the remaining part of this example is to attain a similar functionality using KIS.ME. In particular, the following features should be achieved:

Environment: It is defined within Workshop 1 and employs KIS.BOX 1, hence the traffic lights system presented in Fig. 2.48 is reduced to one KIS.BOX, which changes the colors of its operational LEDs to mimic the behaviour of the traffic lights system.

Triggers: A trigger is associated with pressing KIS.BOX 1 Button 2.

Conditions: The conditions are simply defined by the current state, which is one of those presented in Table 2.7;

Actions: The associated action is simply a consecutive state.

It should be noted that each state described in Table 2.7 is uniquely defined, which makes it possible to form the state-space model using a set of four rules. A sample rule evolving the system from state 1 to state 2 is provided in Fig. 2.49. Moreover,

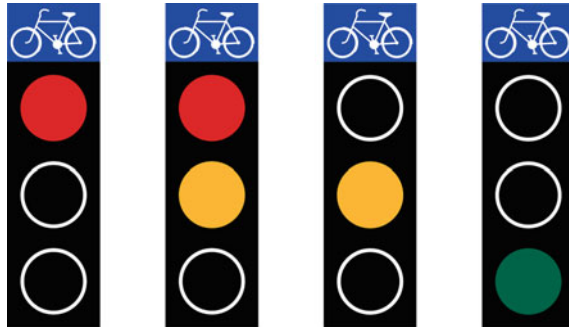


Fig. 2.48 Traffic lights

Table 2.7 KIS.BOX-based states

State	KB operational LED 1	KB operational LED 2
1	Red	Black
2	Red	Yellow
3	Green	Black
4	Green	Yellow

Trigger

KIS.BOX 1 Button 2 Pressed

Conditions

KIS.BOX 1 Button 2 Button 2 Color EQUAL Flashing

Actions

KIS.BOX 1 Set LED Button 2 Color Flashing

Fig. 2.49 Sample traffic lights rule

the initialization of the system requires that KIS.BOX 1 operational LEDs be in one of the quadruple of states defined in Table 2.7. This can be easily achieved using the KIS.BOX digital twin (see Sect. 2.6).

2.11 Mastering Rule Management: Completeness and Consistency

The objective of the two preceding sections was to provide a concise introduction into Rule engine design and the inference mechanism. However, for more complex systems, the number of rules will proliferate. Thus, it is customary to have a tool capable of checking their completeness and consistency. For that purpose, the celebrated decision table [2] is introduced. It operates on Boolean-valued conditions, and hence it is beneficial to recall the essential logical operators provided in their priority order:

- ¬ negation,
- ∧ conjunction,
- ∨ disjunction,
- ⇒ implication,
- ⇔ equivalence.

The behaviour of the above operators is explained in Fig. 2.50. It can be also observed that the implication and equivalence can be expressed by

$$a \Rightarrow b \text{ can be calculated with } \neg a \vee b;$$

$$a \Leftrightarrow b \text{ can be obtained with } (a \Rightarrow b) \wedge (b \Rightarrow a).$$

Therefore, a typical way of expressing a logical implication is IF a THEN b . Thus, according to the truth table for $a \Rightarrow b$, if a is false then it does not matter what b is, and hence the implication is true. Similarly, if a and b are true then the implication is true as well. The last case, i.e., when a is true and b is false, can be explained using the following example:

$$\text{IF } \sin(z) = 0 \text{ THEN } z = 0.$$

Such an implication is false as $z = 0$ is not the only value for which $\sin(z) = 0$. Thus, the implication which is true should be

$$\text{IF } \sin(z) = 0 \text{ THEN } z = k\pi, \text{ with } k \text{ being an integer value.}$$

To summarize these preliminaries, two crucial definitions have to be provided:

Tautology: a statement that is true for every possible interpretation, e.g., (KIS.Box|Status|Operating Mode is Offline) or (KIS.Box|Status|Operating Mode is Online);

Contradiction: a statement that is false for every possible interpretation, e.g., (KIS.Box|Status|Operating Mode is Offline) and (KIS.Box|Status|Operating Mode is Online).

Indeed, it is obvious that in the first case the statement is always true as the KIS.BOX status can be either Offline or Online. Contrarily, it is evident that the second state-

a	$\neg a$
0	1
1	0

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

a	b	$a \Rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

a	b	$a \Leftrightarrow b$
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 2.50 Truth tables

ment is always false as the KIS.BOX status cannot be Offline and Online simultaneously. Thus, it is evident that one should avoid both cases while designing rules with KIS.ME.

2.11.1 Transforming Conditions

A preliminary step for implementing a rule base is to collect all rules and check if it is possible to simplify them. For that purpose, a standard set of transformation strategies can be used:

double negation: $\neg\neg a = a$,

commutativity of conjunction: $a \wedge b = b \wedge a$,

commutativity of disjunction: $a \vee b = b \vee a$,

associativity of conjunction: $(a \wedge b) \wedge c = a \wedge (b \wedge c)$,

associativity of disjunction: $(a \vee b) \vee c = a \vee (b \vee c)$,

distributivity of conjunction: $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$,

distributivity of disjunction: $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$,

idempotency of conjunction: $a \wedge a = a$,

idempotency of disjunction: $a \vee a = a$,

De Morgan's law: $\neg(a \wedge b) = \neg a \vee \neg b$,

De Morgan's law: $\neg(a \vee b) = \neg a \wedge \neg b$,

contraposition law: $a \rightarrow b = \neg b \Rightarrow \neg a$.

Having the above strategies, one can simply associate logical variables with antecedents (see Sect. 2.9). For that purpose, it is suggested to use the following nomenclature:

$$a := \text{LOs EQUAL Value}, \quad (2.3)$$

$$\neg a := \text{LOs NOT Value}. \quad (2.4)$$

Figure 2.51 presents two sample antecedents which can be associated with a logical variable a and its negation $\neg a$. Having such variables, it is easy to write and operate on conditions in a consistent way. Another important aspect pertains to an appropriate use of parentheses. Indeed, due to the operators' priority, $a \vee (b \wedge c)$ can

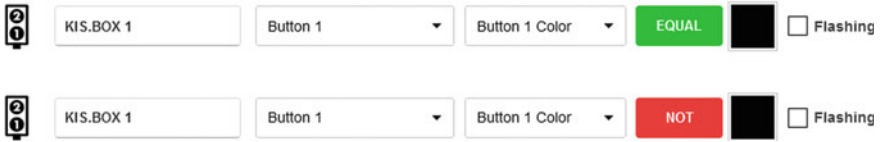


Fig. 2.51 Sample a (top) and $\neg a$ (bottom)

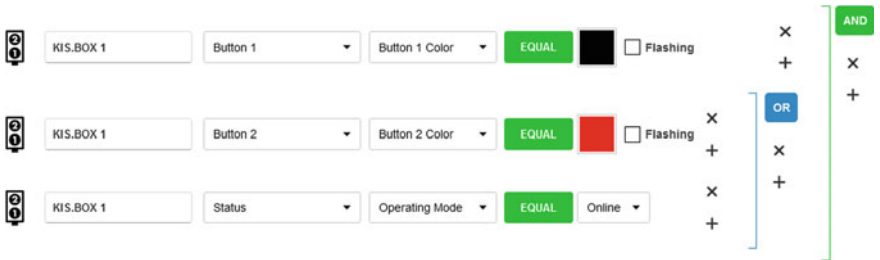


Fig. 2.52 Sample $a \wedge (b \vee c)$ (a : top, b : middle, c : bottom)

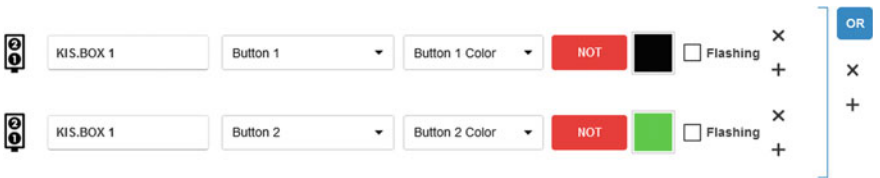


Fig. 2.53 Implementation of $\neg a \vee \neg b$

be simplified to $a \vee b \wedge c$. Contrarily, $a \wedge (b \vee c)$ differs from $a \wedge b \vee c$. This can be clearly observed within Rule engine, which uses different right square brackets for indicating appropriate priorities. Thus, $a \wedge (b \vee c)$ can be exemplified with the conditions portrayed in Fig. 2.52.

Sample simplification

Let us suppose that there are two logical variables defined as

$$a := \text{KIS.BOX 1} \mid \text{Button 1} \mid \text{Button 1 Color EQUAL black,}$$

$$b := \text{KIS.BOX 1} \mid \text{Button 2} \mid \text{Button 2 Color EQUAL green,}$$

and an action must be performed when $\neg(a \wedge b)$ (NAND operation) is true. Unfortunately, such a condition is impossible to implement in Rule engine. However, it is straightforward to observe that by applying De Morgan’s law it is possible to simplify it into $\neg a \vee \neg b$, which can be easily implemented in Rule engine (see Fig. 2.53).

```
(%i17) load(logic);
(%o17) C:/maxima-5.45.1/share/maxima/5.45.1/share/logic/logic.mac

(%i18) logic_simplify( not (a and b));
(%o18) nota or notb
```

Fig. 2.54 Logical expression simplification with Maxima

Table 2.8 Decision table

Condition/action	r_1	r_2	...	r_k
Condition 1	$c_{1,1}$	$c_{1,2}$...	$c_{1,k}$
Condition 2	$c_{2,1}$	$c_{2,2}$...	$c_{1,k}$
⋮	⋮	⋮	...	⋮
Condition n	$c_{n,1}$	$c_{n,2}$...	$c_{n,k}$
Action 1	$a_{1,1}$	$a_{1,2}$...	$a_{1,k}$
Action 2	$a_{2,1}$	$a_{2,2}$...	$a_{2,k}$
⋮	⋮	⋮	...	⋮
Action m	$a_{m,1}$	$a_{m,2}$...	$a_{m,k}$

> Automatic simplification

There are several free and commercial packages which can be used for automatic simplification of logical expressions. Maple [4] (package `Logic`, command `BooleanSimplify`) and Maxima [5] (package `logic`, command `logic_simplify`) are good representative examples of commercial and freely available tools, respectively. Let us employ Maxima for the purpose of simplifying the expression used in the preceding example, i.e., $\neg(a \wedge b)$. The Maxima session implementing this task is presented in Fig. 2.54.

To conclude this section, it should be pointed out that the rule

$$\text{IF antecedent 1 OR antecedent 2 OR } \dots \text{ antecedent } n \text{ THEN Action(s)} \quad (2.5)$$

can be replaced by a set of n rules of the form

$$\text{IF antecedent 1 THEN Action(s),} \quad (2.6)$$

⋮

$$\text{IF antecedent } n \text{ THEN Action(s).} \quad (2.7)$$

2.11.2 Decision Tables

The classical decision table [2, 3, 6, 7] is used to express k rules of the form

$$r_i : \text{ IF Condition 1 and Condition 2, \dots and Condition } n \\ \text{ THEN Action 1 and Action 2, \dots and Action } m, \quad i = 1, \dots, k, \quad (2.8)$$

which can be presented using Table 2.8. The internal horizontal line between conditions is perceived as the and conjunction. Additionally, the double horizontal line separates conditions and actions while the vertical ones distinguish the rules. Thus, any rule r_i can be easily reconstructed from Table 2.8 to (2.8) by reading the column corresponding to r_i in a top-to-bottom order. The entries of the decision table with respect to the conditions, i.e., $c_{i,j}$, in Table 2.8 are as follows:

- T: if the condition must hold;
- F: if the condition does not hold;
- : if the condition is ignored;

while for the actions $a_{i,j}$ we have

- X: if the action has to be executed;
- : if the action has not to be executed.

A set of rules or a decision table have the following important features:

Redundancy: If there is a situation in which conditions of two rules with the same actions hold, then they are called redundant ones.

Inconsistency: If there is a situation in which conditions of two rules with different actions hold, then they are called inconsistent ones.

Completeness: For every situation there is a rule whose conditions will be satisfied.

Checking redundancy and inconsistency

Let us consider three conditions which have to be implemented using Rule engine within Workshop 1 with KIS.BOX 1 and KIS.LIGHT 0:

Condition 1: KIS.BOX 1 | Button 1 | Button 1 Color is black;

Condition 2: KIS.BOX 1 | Button 2 | Button 2 Color is blue;

Condition 3: KIS.LIGHT 0 | LED Color | Color is green.

There are also two actions:

Action 1: KIS.LIGHT 0 | Set LED | LED Color | is black;

Action 2: KIS.LIGHT 0 | Set LED | LED Color | is blue.

Let us suppose that the above conditions and actions were used to implement three rules, r_1 , r_2 and r_3 , expressed in the form of the decision table detailed in Table 2.9. Let us consider a situation in which Condition 1 is T (true) while Condition 2 and

Table 2.9 Decision table with redundancy and inconsistency

Condition/Action	r_1	r_2	r_3
Condition 1	T	–	T
Condition 2	–	F	T
Condition 3	F	F	–
Action 1	X	X	–
Action 2	–	–	X

Table 2.10 Decision table with inconsistency

Condition/Action	r_1^1	r_3
Condition 1	–	T
Condition 2	–	T
Condition 3	F	–
Action 1	X	–
Action 2	–	X

Condition 3 are F (false). In such a case both rules r_1 and r_2 are active. Since they have identical actions (Action 1), they are redundant, which means that they can be merged into one equivalent rule r_1^1 . The resulting decision table is presented in Table 2.10. Let us consider a situation in which Condition 1 and Condition 2 are T (true) while Condition 3 is F (false). It can be easily observed that rules r_1^1 and r_3 are inconsistent because their condition sets are satisfied while they have different sets of actions. The inconsistent rules denote the situation in which different things may happen under the same circumstances. Indeed, two contradictory actions will be initiated:

Action 1: KIS.LIGHT 0 | Set LED | LED Color | is black;

Action 2: KIS.LIGHT 0 | Set LED | LED Color | is blue.

To summarize, a simple rule reduction principle can be stated as below.

➤ Rule reduction principle

If there are two rules l and s with the same actions and identical condition entries $c_{i,l}$ and $c_{i,s}$ except for $c_{j,l} \neq c_{j,s}$, then they are replaced with a single new rule f with a condition entry $c_{j,f}$ equal to “–”. Note that as “–” represents T/F, it should be perceived as equal to both T and F.

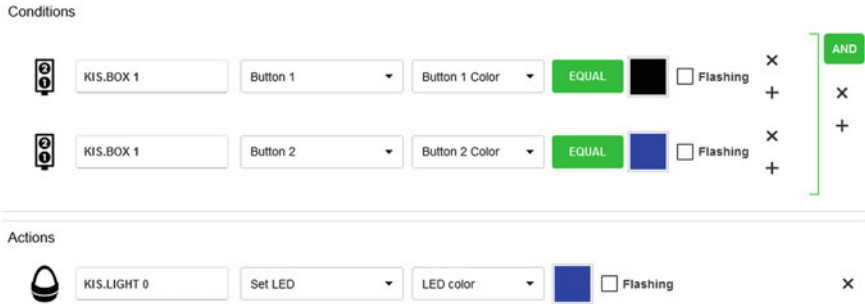


Fig. 2.55 Rule r_3 implemented within Rule engine

> **Implementing decision tables with Rule engine**

Implementation of decision tables within Rule engine can be easily realized using the following procedure:

1. Select the i -th rule.
2. Read the rule in order from top-to-bottom and perform the following translation:
 - if an entry is equal to “T”, then introduce the condition using (2.3);
 - if an entry is equal to “F”, then introduce the condition using (2.4);
 - if an entry is equal to “–”, then ignore it.

As an example, let us consider rule r_3 in Table 2.10, which is exemplified in Fig. 2.55.

The above implementation strategy is applicable to a set of simple conditions in the form of either (2.3) or (2.4). However, it can be easily extended to more advanced structures. On the other hand, the logical expressions can be simplified using the strategies proposed in the preceding section (see, e.g., (2.5) and (2.7)) or transformed into conjunctive normal form (see, e.g., [2] for a comprehensive explanation).

> **Rule base completeness**

Having a way of checking the redundancy and consistency, it is possible to provide a strategy for verifying the completeness of a set of rules. Since each condition in a decision table (Table 2.8) is a Boolean-valued one, this simply means that the complete number of rules is equal to

$$k = 2^n, \tag{2.9}$$

where k is the total number of rules while n is the number of conditions (cf. (2.8)). Thus, a rule should be defined for every possible situation. As an example, let us consider the decision table presented in Table 2.10. Thus, for three conditions one can easily see that (2.9) implies that there should be $k = 8$ rules. Contrarily, there are two rules in Table 2.10. However, due to the use of “-”, rule r_1^1 may have four alternative forms:

$$r_1^1 \in \left\{ \begin{bmatrix} F \\ F \\ F \end{bmatrix}, \begin{bmatrix} T \\ T \\ F \end{bmatrix}, \begin{bmatrix} F \\ T \\ F \end{bmatrix}, \begin{bmatrix} T \\ F \\ F \end{bmatrix} \right\},$$

while rule r_2 has two alternative ones:

$$r_2 \in \left\{ \begin{bmatrix} T \\ T \\ F \end{bmatrix}, \begin{bmatrix} T \\ T \\ T \end{bmatrix} \right\}.$$

This clearly means that there are six rules and the decision table (Table 2.10), and hence the set of rules is incomplete. Thus, it is possible to verify (2.9) with

$$k_r = \sum_{i=1}^{n_r} 2^{n_{i,-}}, \quad (2.10)$$

where n_r stands for the number of rules in Table 2.8 while $n_{i,-}$ is the number of instances of “-” in the i -th rule, $i = 1, \dots, n_r$. In the example presented in Table 2.8, one can easily identify what follows

- there are two rules $n_r = 2$;
- there are two instances of “-” in rule r_1^1 , which results in $n_{1,-} = 2$;
- there is one “-” in rule r_2 , i.e., $n_{2,-} = 1$;
- thus, (2.10) implies that $k_r = 2^2 + 2^1 = 6$, which clearly means that $k_r < k$ and the set of rules is incomplete.

It should be pointed out that there are situations in which rule base completeness is not necessary. Indeed, if it is guaranteed that not all possible situations pertain to input variables, and hence, conditions are possible, then the number of rules can be smaller. Such a situation may occur during the state-space modelling presented in Sect. 2.10. In such a case, the consecutive states are cyclically realized, which implies appropriate rule order execution. Finally, it should be pointed out that triggers may also have influence on the final number of rules. Indeed, they cause that some rules are not fired for a given trigger setting.

2.12 Case Study: Trend Plotting and Performance Analysis

The objective of the preceding three sections was to introduce Rule engine, which makes it possible to bring a given system alive according to a predefined set of rules. Having such features, it is possible to monitor KIS.Device performance with Datapoints and the associated widgets described in Sect. 2.7. For that purpose, let us reconsider the traffic lights example presented in Sect. 2.10. Let us start with transforming Table 2.7 into a decision table assuming that the initial state of the system is (cf. Sect. 2.6)

```
KIS.BOX 1 | Button 1 |Button 1 Color is red;
KIS.BOX 1 | Button 2 |Button 2 Color is black.
```


By observing an obvious condition stating that an operational LED cannot have two different colors simultaneously, the resulting decision table is given in Table 2.11. This implies that there is no need for implementing “F”-valued entries, which simplifies the Rule engine structure. For `Button 1 Color`, T is equivalent to red while F stands for green. Similarly, for `Button 2 Color`, T is equivalent to black while F stands for yellow. It is straightforward to observed that there are two conditions, and hence (2.9) implies that a complete set of rules should have four rules. This is exactly the case. Moreover, the rules are consistent because each of them pertains to a different set of actions. Finally, a Rule engine-based implementation of r_1 – r_4 is provided in Figs. 2.56, 2.57, 2.58 and 2.59. Having a fully functional system, it is possible to design a dashboard containing

- a floorplan within Workshop 1 (cf. Sect. 2.8 for guidelines),
- a digital twin of KIS.BOX 1 implementing the traffic lights system,
- a Datapoint Chart widget containing two (cf. Sect. 2.7 for details) plots showing the color of KIS.BOX first and second operational LEDs. This can be achieved using



```
button1ColorKpiDuration,
button2ColorKpiDuration.
```

The resulting dashboard is presented in Fig. 2.60. Such a design allows intuitive visualization and analysis of the behaviour of the traffic lights system. Indeed, the plots in Fig. 2.61 correspond to the operational LED colors of the first (green) and the second (yellow) buttons. As can be observed, both of them have two possible values only, which can be recorded and analysed over a specified time period. In particular, the green line switches between the red (5) and the green (3) level. Similarly, the yellow line switches between black (2) and yellow (7).

Trigger

 KIS.BOX 1 Button 2 Pressed

Conditions

 KIS.BOX 1 Button 2 Button 2 Color EQUAL  Flashing

Actions






 KIS.BOX 1 Set LED Button 2 Color  Flashing

Fig. 2.56 Traffic lights rule r_1 implemented within rule engine

Trigger

 KIS.BOX 1 Button 2 Pressed

Conditions

 KIS.BOX 1 Button 2 Button 2 Color NOT  Flashing

Actions







 KIS.BOX 1 Set LED Button 2 Color  Flashing

Fig. 2.57 Traffic lights rule r_2 implemented within rule engine

Conditions

 KIS.BOX 1 Button 1 Button 1 Color EQUAL  Flashing

 KIS.BOX 1 Button 2 Button 2 Color NOT  Flashing

Actions



 KIS.BOX 1 Set LED Button 1 Color  Flashing

Fig. 2.58 Traffic lights rule r_3 implemented within rule engine

Conditions

	KIS.BOX 1	Button 1	Button 1 Color	NOT		<input type="checkbox"/> Flashing
	KIS.BOX 1	Button 2	Button 2 Color	NOT		<input type="checkbox"/> Flashing

Actions

	KIS.BOX 1	Set LED	Button 1 Color		<input type="checkbox"/> Flashing
--	-----------	---------	----------------	--	-----------------------------------

Fig. 2.59 Traffic lights rule r_4 implemented within rule engine

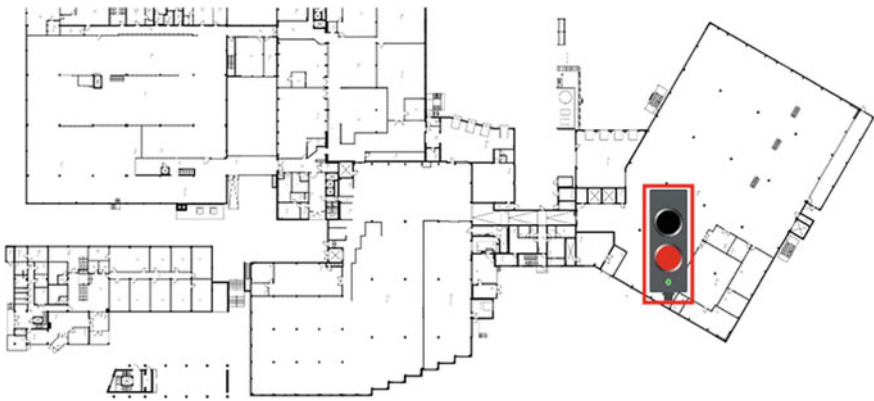


Fig. 2.60 Traffic lights dashboard

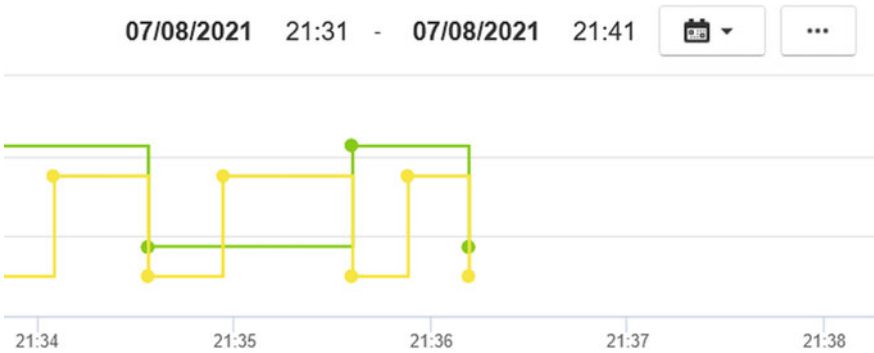


Fig. 2.61 Traffic lights datapoint chart

Table 2.11 Decision table for traffic lights

Condition/Action	r_1	r_2	r_3	r_4
KIS.BOX 1 Button 1 Button 1 Color	–	–	T	F
KIS.BOX 1 Button 2 Button 2 Color	T	F	F	F
KIS.BOX 1 Set LED Button 1 color is red	–	–	–	X
KIS.BOX 1 Set LED Button 1 color is green	–	–	X	–
KIS.BOX 1 Set LED Button 2 color is black	–	X	–	–
KIS.BOX 1 Set LED Button 2 color is yellow	X	–	–	–

2.13 Training Exercises

The objective of this section is to provide a set of practical exercises which can be used for validating the knowledge and skills gathered in this chapter. It is assumed that the user realizing these exercises has appropriate rights and permissions. This can be easily verified using Table 2.5.

2.1 Asset onboarding and management

1. Check the availability/feasibility of the following points:
 - a KIS.Device,
 - an M12-to-USB cable (see Table 2.4),
 - a computer/tablet equipped with a web browser and a USB port,
 - a company account with admin user rights (see Sect. 2.3),
 - WLAN access with permission credentials.
2. Go to <https://kismanager.rafi.de> and log in with user company credentials.
3. Perform the onboarding procedure of KIS.Device according to the guidelines presented in Sect. 2.1.
4. Depending on the KIS.Device type, change the name of the device to either KIS.BOX YOUR NAME or KIS.LIGHT YOUR NAME (see Sect. 2.4, Fig. 2.18).
5. Assign the KIS.Device to Workshop 1 and Workshop 2.

2.2 Creating a new user

Using an arbitrary e-mail address at your disposal, perform the following:

1. Create a user according to the guidelines presented in Sect. 2.3.1.
2. Give the new user installer rights (see Table 2.3).
3. Assign the new user to Workspace 1 and Workspace 2.

2.3 Dashboard and digital twin design

Exercise requirements: The exercise requires access to one KIS.BOX and one KIS.LIGHT. Proceed according to the following tasks:

1. Go to Main menu → Assets.
2. Select KIS.BOX, proceed to its dashboard and modify its name to “Sample dashboard”.
3. Design a KIS.BOX digital twin.
4. Add Datapoint Chart to the dashboard capable of displaying Datapoints:

```
button1ColorKpiDuration,  
button2ColorKpiDuration.
```

5. Using the digital twin, change arbitrarily the colors of both the first and the second operational LEDs.
6. Record the behaviour in the Datapoint Chart and store it with a CSV file.
7. Open the CSV file in a MS Excel-like software and compare its content with Table 2.2.
8. Use the Time drive feature of the dashboard and observe KIS.BOX historical behaviour.
9. Repeat, analogously, points 1–8 with KIS.LIGHT.

2.4 Floorplan widget

Exercise requirements: The exercise requires access to one KIS.BOX and one KIS.LIGHT assigned to a selected workspace.

1. Prepare your own floorplan using vector graphic software (e.g., Inkscape [1]) and save it as an SVG file.
2. Add the Floorplan widget to Workspace and locate KIS.LIGHT and KIS.BOX on it.

2.5 KIS.LIGHT ruling

Exercise requirements: completed Exc. 2.4.

1. Write a rule called `rule b2r`:

```
Triggers: KIS.LIGHT operational LED is black;  
Conditions: KIS.LIGHT operational LED is black;  
Actions: KIS.LIGHT operational LED is red.
```

2. Write a rule called `rule_r2b`:

Triggers: KIS.LIGHT operational LED is red;

Conditions: KIS.LIGHT operational LED is red;

Actions: KIS.LIGHT operational LED is black.

3. Go to Main menu → Assets, select KIS.LIGHT.
4. Using the KIS.LIGHT digital twin, set its operational LED color to black.
5. Add a Datapoint Chart associated with KIS.LIGHT to the Workspace dashboard, and attach its plot to `led1ColorKpiDuration`.
6. Fill in the rest of Datapoint Chart configuration parameters in an arbitrary way.
7. Using the Zoom In/Out feature, observe the behaviour of the switching signal within the last two minute.
8. What can you say about the switching frequency/period? Is it uniform?

2.6 KIS.LIGHT ruling continued

Exercise requirements: completed Exc. 2.5. Using a similar strategy like in Exc. 2.5, perform the following:

1. Write a set of rules enabling KIS.LIGHT transition with consecutive states described in Table 2.2, i.e., Blue, Turquoise, Black, Green, Magenta, Red, White, Yellow.
2. Using the Zoom In/Out feature, observe the behaviour of the switching signal within the last two minutes.
3. What can you say about the transition periods?

2.7 KIS.BOX traffic lights

Exercise requirements: The exercise requires access to one KIS.BOX assigned to a selected workspace.

1. Read and analyse the traffic light case study described in Sect. 2.12.
2. Build your own traffic light system according to the strategy described in Sect. 2.12.
3. Analyse the transition periods. What can you say about them?

2.8 Rule simplification I

1. Analyse the rule transformation and simplification strategies presented in Sect. 2.11.1; in particular, automatic simplification using the Maxima software, which is presented in Fig. 2.54.
2. Install the freely available Maxima [5] software.
3. Repeat the automatic simplification process presented in Fig. 2.54.
4. Let a be a logical variable denoting the fact that KIS.BOX is online. Thus, $\neg a$ signifies the fact that it is offline:
 - What can you say about $a \wedge (\neg a)$?
 - What can you say about $a \vee (\neg a)$?

2.9 Rule simplification II

Exercise requirements: The exercise requires access to one KIS.BOX and KIS.LIGHT assigned to a selected workspace.

1. Let us define the following logical variables:

$$\begin{aligned} a &:= \text{KIS.BOX} \mid \text{Button 1} \mid \text{Button 1 Color EQUAL red,} \\ b &:= \text{KIS.BOX} \mid \text{Button 2} \mid \text{Button 2 Color EQUAL black,} \\ c &:= \text{KIS.LIGHT} \mid \text{LED} \mid \text{LED Color EQUAL green.} \end{aligned}$$

2. Implement a rule with the following condition:

$$(a \wedge b \wedge c) \vee (\neg a) \vee (a \wedge (\neg b) \wedge c), \quad (2.11)$$

taking as a trigger the pressing of the first KIS.BOX button event along with an action:

Action: KIS.LIGHT 0 | Set LED | LED Color | is red.

Hint: Use Maxima to simplify the above logical expression.

3. What can you say about the usage of variable b ?

2.10 Decision tables

Exercise requirements: The exercise requires access to one KIS.BOX and KIS.LIGHT assigned to a selected workspace.

1. Let us define three conditions:

Condition 1: KIS.BOX | Button 1 | Button 1 Color EQUAL red,
 Condition 2: KIS.BOX | Button 2 | Button 2 Color EQUAL black,
 Condition 3: KIS.LIGHT | LED | LED Color EQUAL green;

along with two actions:

Action 1: KIS.LIGHT 0 | Set LED | LED Color | is red;

Action 2: KIS.BOX | Set LED | Button 1 color | is blue;

and a trigger associated with pressing the first KIS.BOX button event.

2. The rules involving the above conditions and actions were initially developed and described using the decision table given in Table 2.12.
3. Simplify the decision table into a new one with four rules only.
4. Check completeness of the obtained decision table.
5. Implement the obtained decision table with Rule engine.

2.11 Battery assembly system

Exercise requirements: The exercise requires access to one KIS.BOX assigned to a selected workspace.

Table 2.12 Initial decision table

Condition/ Action	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
Condition 1	T	F	T	F	T	F	T	F
Condition 2	T	T	F	T	F	F	T	F
Condition 3	T	T	F	F	T	F	F	T
Action 1	–	X	–	–	–	X	–	X
Action 2	X	–	X	–	X	X	X	–

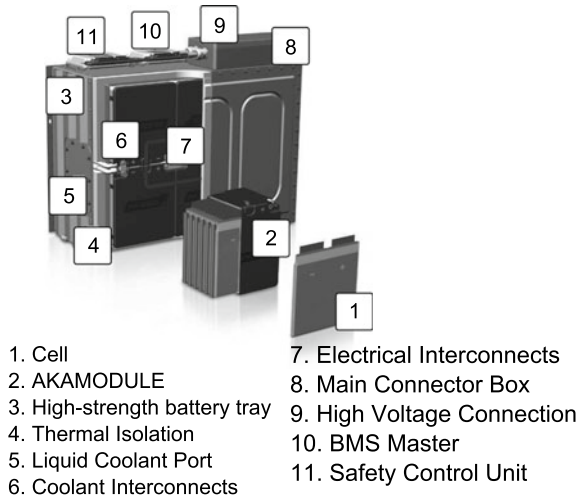
Table 2.13 KIS.BOX states

State	KB Button 1 color	KB Button 2 color	Action
1	Blue	Blue	Can start
2	Red	Blue	Cell mounting
3	Green	Blue	Cell mounting completed
4	Green	Red	Cell-controller linking
5	Green	Green	Mounting completed

1. Let us consider the battery system presented in Fig. 2.62.
Such a system undergoes manual assembly, which, in simplified form, can be described by two tasks:
 - a. cell mounting,
 - b. controller linking.

Battery mounting should be realized according to the cyclically repeated states described in Table 2.13. Note that the initial state should be State 1.
2. Prepare and simplify a decision table pertaining to the states described in Table 2.13.
3. Check completeness and consistency of the obtained decision table.
4. Form a digital twin of KIS.BOX as well as an arbitrary floorplan containing selected parts of Fig. 2.62 integrated with KIS.BOX.
5. Use the obtained decision table and implement it with Rule engine.
6. With the KIS.BOX digital twin, set the initial KIS.BOX state to State 1 and check if the system performs correctly (cf. Table 2.13).
7. Add the Datapoint Chart widget to the Workspace dashboard, which makes it possible to analyse the colors of the operational LEDs of both the first and the second KIS.BOX button.
8. analyse both the historical and the current behaviour of the system. A hint: You can also use the time drive feature of the dashboard.

Fig. 2.62 Battery assembly



2.14 Concluding Remarks

The objective of this chapter was to provide a self-contained introduction to the KIS.ME IoT platform. In particular, preliminary information about both the hardware and software layers was introduced along with suitable operating procedures for accessing KIS.MANAGER and onboarding the hardware layer. Subsequently, a hierarchical KIS.ME structure was presented, which associates assets, users and workspaces along with suitable rights and permissions. For that purpose, a systematic set of guidelines concerning users, assets and workspace management was provided. Such essential knowledge enabled introduction of dashboards and widgets, which form the basis for the KIS.ME HMI interface. Concerning the widgets, particular attention was focused on hardware digital twins as well as floorplans exemplifying real life systems with an integrated set of assets. Such a couple was further extended with the Datapoints chart enabling graphical visualization of system performance. The rest of the chapter was devoted to system management using Rule engine. In particular, it started with a concise introduction to the graphical rule building structure. Subsequently, a state-space modelling strategy was proposed, which guarantees cyclical behaviour of the system. Finally, more advanced techniques for managing a set of rules were introduced, which allow their simplification as well as completeness and consistency verification. The chapter was concluded with a series of practical exercises, which certify the knowledge provided within it.

References

1. Inkscape. <https://inkscape.org/>. Accessed 25 June 2021
2. A. Ligeza, *Logical Foundations for Rule-Based Systems* (Springer, Berlin, 2006)
3. C. Grosan, A. Abraham, Rule-based expert systems, in *Intelligent Systems* (Springer, Berlin, 2011), pp.149–185
4. Maple. <https://www.maplesoft.com/>. Accessed 15 July 2021
5. Maxima. <https://maxima.sourceforge.io/>. Accessed 15 July 2021
6. F. Alsolami, M. Azad, I. Chikalov, M. Moshkov, *Decision and Inhibitory Trees and Rules for Decision Tables with Many-valued Decisions* (Springer, Berlin, 2019)
7. J.R. Metzner, B.H. Barnes, *Decision Table Languages and Systems* (Academic Press, New York, 2014)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 3

Towards Logistic Applications



3.1 Access Control

Irrespective of the application being considered, one of the important features while operating KIS.Devices is to guarantee the desired access control. This simply means that one should provide means for preventing the situation in which persons without appropriate access permissions operate KIS.BOXes installed in the system. In other words, even if they touch the buttons of a given KIS.BOX, there should be no effect associated with such an action. In this section, two kinds of access control are considered:

General access control: each user with a granted permission is treated in the same way in the system;

Individual access control: users with a granted permission are individually identified in the system.

Irrespective of the selected access control strategy, it is proposed to use the RFID (radio-frequency identification) option as a user identification tool. However, before proceeding to the details, let us provide an introductory example involving KIS.BOX GPIO.

KIS.BOX with a photoelectric sensor

Let us consider a normally open photoelectric sensor, which is connected with KIS.BOX according to the scheme presented in Fig. 3.1. There are several such sensors with light-based binary switching properties. Similarly to KIS.BOX (see Sect. 2.1), they also use an M12 connection as well as a 24 V power supply. Such a sensor works as relay, which simply changes the KIS.BOX digital input value depending on the light condition. With a hardware infrastructure, it is possible to implement sample rules reacting to the changes of the photoelectric sensor.

Environment: It is defined within Workshop 1 and employs KIS.BOX 1. There are two rules which are determined using the settings presented below.

Triggers: There are two rules and their triggers are associated with pressing either KIS.BOX 1 Button 1 or KIS.BOX 2 Button 2.

Conditions: If the sensor is exposed to the light, then a logical false state (Off) of Input 1 is obtained. Contrarily, if there is no light, then a logical true state (On) is generated. Having this in mind, the idea is to equip the above-defined rules with a preliminary condition associated with the logical false state (Off) of Input 1. Additionally, the first rule will check if the KIS.BOX 1 operational LED 1 color is red while the second one will verify if the KIS.BOX 1 operational LED 1 color is blue.

Actions The action associated with the first rule causes that the KIS.BOX operational LED 1 color switches to blue. The second rule performs in a similar way, but KIS.BOX1 operational LED 1 transforms into red.

Finally, it should be indicated that the initial state of KIS.BOX operational LED 1 is red. This can be easily realized with the KIS.BOX 1 digital twin (cf. Sect. 2.6). The implementation of the above-defined two rules is presented in Figs. 3.2 and 3.3. As a result of the above hardware–software configuration, a useful light-based access control system is developed. Indeed, if the photoelectric sensor is not exposed to the light, then no action is performed.

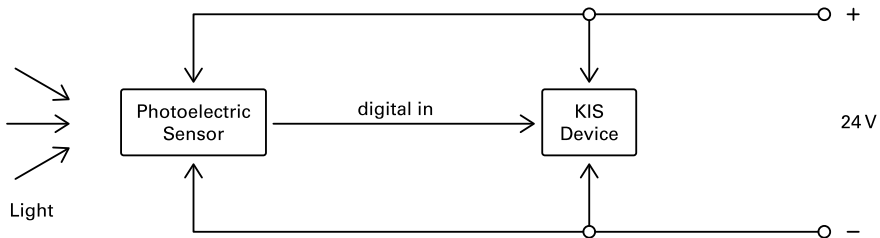


Fig. 3.1 KIS.BOX with a photoelectric sensor

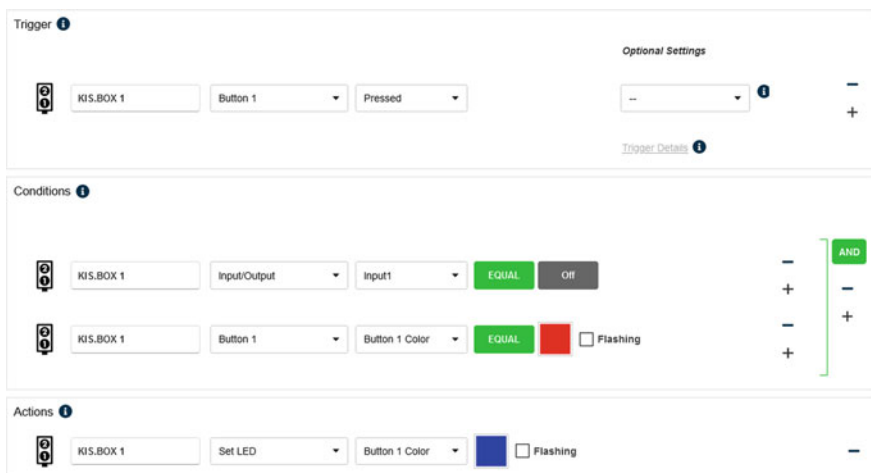


Fig. 3.2 First rule for KIS.BOX with a photoelectric sensor

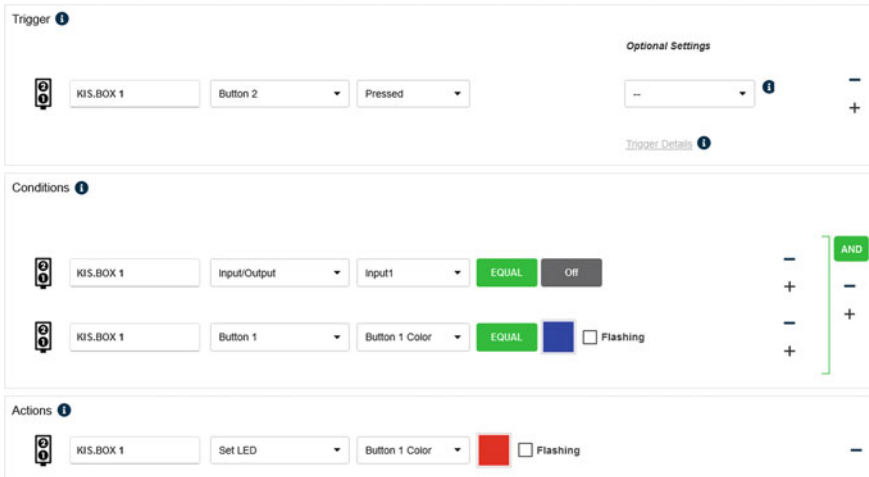


Fig. 3.3 Second rule for KIS.BOX with a photoelectric sensor



Fig. 3.4 RAMO product line

Remark 3.1 The above system configuration can be adapted to a wide spectrum of input devices and sensors. For example, a RAMO product line can be efficiently used for that purpose (see Fig. 3.4).

Following such a preliminary access control example, it is possible to go back to access control with RFID readers. Irrespective of the access control strategy being

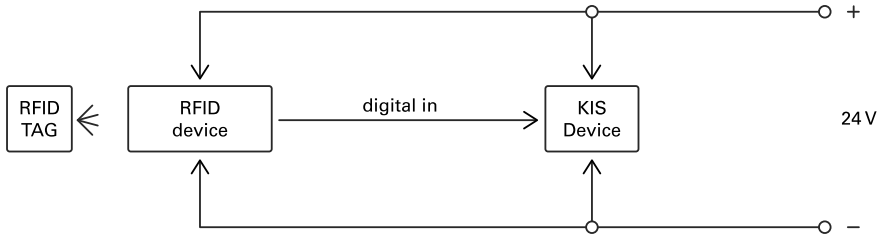


Fig. 3.5 RFID-based access control scheme

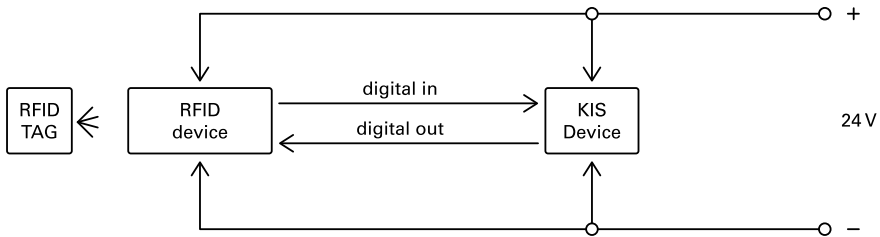


Fig. 3.6 RFID-based access control scheme with feedback from KIS.MANAGER

used (general or individual), the general scheme of the proposed approach is given in Fig. 3.5. It consists of a KIS.Device and an RFID reader including or associated with a processing unit, which, after reading an appropriate RFID tag, may perform the following commands:

Reset command: It sends a false (Off)–true (On)–false (Off) sequence to a KIS.Device;

Identification command: It sends a desired series of false (Off)–true (On)–false (Off) values, which identifies a user.

Let us start with the general access control strategy. The simplest approach which can be employed in this case can be realized in a similar way as the one portrayed in Fig. 3.5. If an appropriate RFID tag is recognized, then the RFID reader (possibly equipped with a processing unit) simply sends an identification command consisting of the following sequence: false–true–false (Off–On–Off). As a consequence, the desired digital input state of the KIS.Device is changed–accordingly. The rest of the configuration has to be realized in KIS.MANAGER. For that purpose, the digital output of the KIS.Device can be employed, which alternates between true (On) and false (Off). This corresponds to two possible situations, i.e., access granted (On) or access denied (Off). An appealing property of such a solution is that, apart from KIS.MANAGER, information about the current access control status can be sent to external hardware / software, which is visualized in Fig. 3.6.

Access control with KIS.BOX

Let us consider the scheme presented in Fig. 3.6, with the KIS.Device being KIS.BOX 1 operating within Workshop 1. Moreover, for the sake of communication purposes, digital Input 1 and Output 1 are utilized. For such a configuration, the resulting access control rules are given in Figs. 3.7 and 3.8. Finally, any access control-dependent rule can simply use KIS.BOX 1 Output 1 to identify the current access control status. Note also that the above solution can also be easily realised with KIS.LIGHT.

The screenshot shows a rule configuration interface with three main sections: Trigger, Conditions, and Actions. Each section has a small icon with a question mark and a number 1.

- Trigger:** KIS.BOX 1, Input/Output, Input1, On (High). Optional Settings: --.
- Conditions:** KIS.BOX 1, Input/Output, Output1, EQUAL, Off.
- Actions:** KIS.BOX 1, Set Input/Output, Output1, On (High).

Fig. 3.7 Rule for granting access

The screenshot shows a rule configuration interface with three main sections: Trigger, Conditions, and Actions. Each section has a small icon with a question mark and a number 1.

- Trigger:** KIS.BOX 1, Input/Output, Input1, On (High). Optional Settings: --.
- Conditions:** KIS.BOX 1, Input/Output, Output1, EQUAL, On (High).
- Actions:** KIS.BOX 1, Set Input/Output, Output1, Off.

Fig. 3.8 Rule for denying access

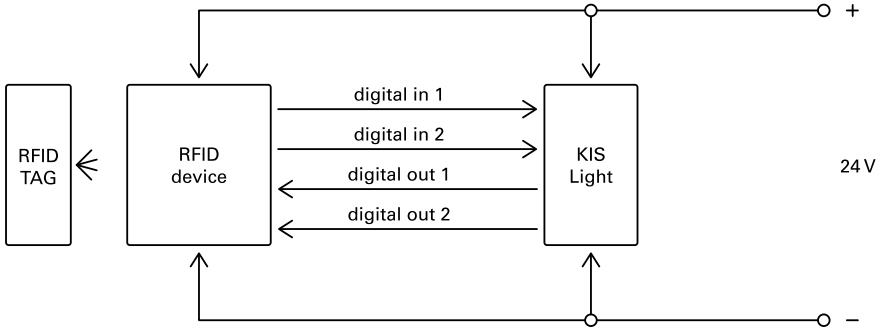


Fig. 3.9 RFID-based individual access control scheme with feedback from KIS.MANAGER

Having a strategy for general access control, which is presented in Fig. 3.9, the proposed individual access control obeys the following laws:

1. Each user has its own unique identification command, i.e., a predefined series of true (On)–false (Off) values.
2. KIS.LIGHT is used for user identification purposes, i.e., each KIS.LIGHT operational LED color (apart from black) is used to identify a given user.

A natural constraint of such an approach is associated with the number of available colors indicated in Table 2.2. Thus, if black is excluded, then there are seven colors applicable. However, such a set of available users is usually sufficient for most practical applications. On the other hand, this limitation can be easily tackled by utilizing, e.g., two KIS.LIGHTs working together. Finally, the list of users along with their identification colors and commands are given in Table 3.1. Note that, for the sake of simplicity, the On and Off states are denoted by 1 and 0, respectively. Having all the above ingredients, let us provide the final operational procedure:

Step 0: The KIS.LIGHT operational LED color is set to black and Output 2 is true (On).

Step 1: If the i -th user's RFID tag is recognized and the selected KIS.LIGHT Output 2 is true (On), then the RFID-based device sends sequentially the i -th identification command to Input 1 according to the following procedure:

- the state of Input 1 is fed to Output 1;
- the KIS.LIGHT color is changed according to the currently received command (cf. Table 3.1);
- the state of Output 1 is read by the RFID-based device.

If a complete i th user identification command is received by the RFID-based device, then go to Step 3.

Step 2: If i -th user RFID tag is recognized and the selected KIS.LIGHT 0 Output 2 is false (Off), then go to Step 3.

Table 3.1 Predefined user colors and commands

User	Color	Identification command
1	Blue	010
2	Turquoise	01010
3	Green	0101010
4	Magenta	010101010
5	Red	01010101010
6	White	0101010101010
7	Yellow	010101010101010

Step 3: The RFID-based device sends a reset command to KIS.LIGHT Input 2 and then the status of Output 2 is changed to the opposite logical value. Moreover, if Output 2 is true (On), then set the KIS.LIGHT 0 operational LED color to black.

Let us start the analysis by noting that Step 0 is performed only once after completing the design of the access control system. Similarly, as in the general access control case, the proposed strategy alternates between two phases, which correspond to granted and denied access. The alternation can be performed by authorized users exclusively. Note also that the color assignment presented in Table 3.1 can be freely modified, while the identification commands should remain as proposed. This will prevent ambiguities when identifying users.

KIS.LIGHT-based individual access control for three users

For presentation purposes, let us consider a set of three users presented in Table 3.2. The proposed strategy is to be implemented using KIS.LIGHT 0 operating within Workshop 1. The implementation starts with Step 0, which pertains to the initial KIS.LIGHT conditions:

- the KIS.LIGHT operational LED color is black;
- KIS.LIGHT digital Output 2 is Off.

Table 3.2 Three users' colors and commands

User	Color	Integer value
1	Blue	010
2	Turquoise	01010
3	Green	0101010

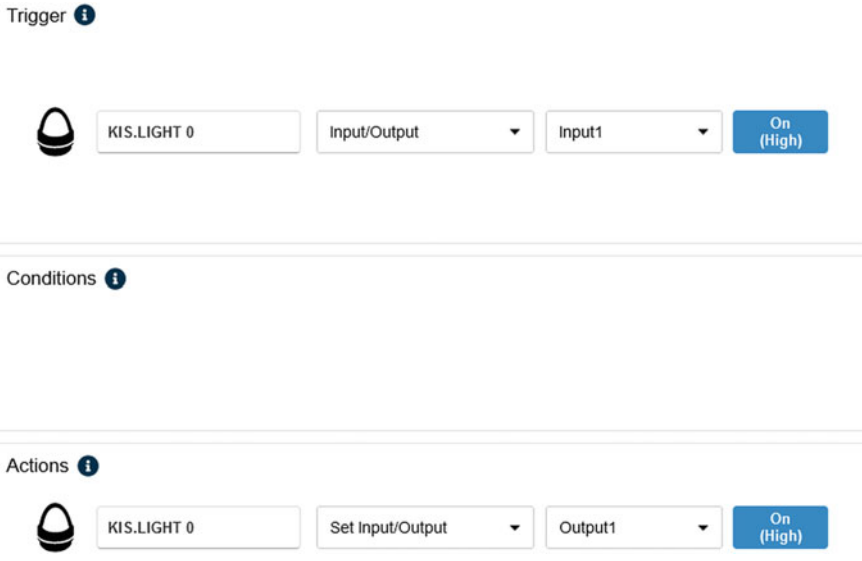


Fig. 3.10 Rule for transferring the on state between input 1 and output 1

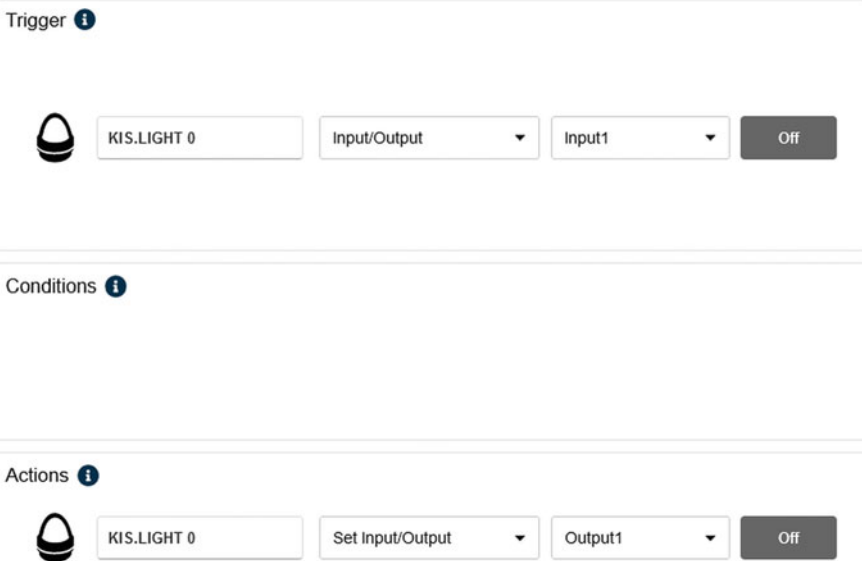


Fig. 3.11 Rule for transferring the off state between input 1 and output 1

The above step is realized only once and can be easily performed using the KIS.LIGHT digital twin (see Sect. 2.6) and/or a dedicated rule performing such an action. Now, let us proceed to Step 1. The first point of this step pertains to transferring Input 1 to Output 1. This task is realized with the two rules presented in Figs. 3.10 and 3.11. The second point of Step 1 concerns changing the KIS.LIGHT

operational LED color according to the currently received identification sequence. It is clear from Table 3.2 that there should be three rules governing such a transition. Thus, for a particular user, the transitions, i.e., answers to the identification command, should be as follows:

- User 1: Blue,
- User 2: Blue→Turquoise,
- User 3: Blue→Turquoise→Green.

To tackle this problem, the rules presented in Figs. 3.12, 3.13 and 3.14 are implemented. Subsequently, the complete Step 2 is realized by the RFID-based device, while by proceeding to Step 3 one can easily observe that the rules alternating the logical state of Output 2 depending on Input 2 can be realized in an analogous way as those presented in Figs. 3.7 and 3.8. Thus, they are simply omitted. Finally, by proceeding to Step 3 one can observe that there is only one rule which has to be carried out, i.e., if Output 2 is true (On), then set the KIS.LIGHT 0 operational LED color to black. This rule is implemented according to Fig. 3.15. Its implementation completes the entire design procedure and the individual access control scheme is ready to use. Finally, it should be pointed out that the extension to seven users presented in Table 3.1 is straightforward and requires four additional rules only, i.e., those for users 4–7.

The screenshot shows a configuration interface for a rule. It is organized into three main sections: Trigger, Conditions, and Actions. Each section starts with a 'Trigger' icon and a title with an information icon.

- Trigger:** The trigger is 'KIS.LIGHT 0'. The condition is 'Input/Output' set to 'Input1' and the state is 'On (High)'. There are 'Optional Settings' and 'Trigger Details' links.
- Conditions:** The condition is 'KIS.LIGHT 0' LED color is 'EQUAL' to black. There is a 'Flashing' checkbox which is unchecked.
- Actions:** The action is 'Set LED' to blue. There is a 'Flashing' checkbox which is unchecked.

Fig. 3.12 Rule for changing the KIS.LIGHT operational LED color from black to blue

Trigger ⓘ

Optional Settings

KIS.LIGHT 0 Input/Output Input1 On (High) -- ⓘ

[Trigger Details](#) ⓘ

Conditions ⓘ

KIS.LIGHT 0 LED LED color EQUAL Flashing

Actions ⓘ

KIS.LIGHT 0 Set LED LED color Flashing

Fig. 3.13 Rule for changing the KIS.LIGHT operational LED color from blue to turquoise

Trigger ⓘ

Optional Settings

KIS.LIGHT 0 Input/Output Input1 On (High) -- ⓘ

[Trigger Details](#) ⓘ

Conditions ⓘ

KIS.LIGHT 0 LED LED color EQUAL Flashing

Actions ⓘ

KIS.LIGHT 0 Set LED LED color Flashing

Fig. 3.14 Rule for changing the KIS.LIGHT operational LED color from turquoise to green

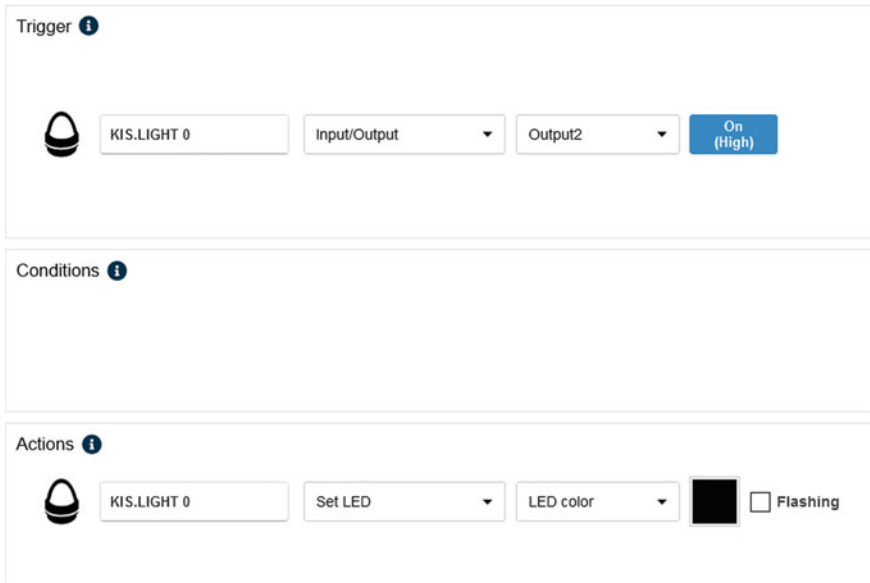


Fig. 3.15 Rule for changing the KIS.LIGHT 0 operational LED color to black

3.1.1 Managing a Small Warehouse

The warehouse management system design problem can be settled using different approaches and strategies [1]. Irrespective of the hardware and software structure being used, such systems aim at improving performance efficiency and minimizing overall costs. Thus, one should find a balance between the expense of implementing such warehouse management systems and the potential overall cost minimization.

The objective of this section is to provide guidelines for implementing a straightforward strategy for managing a small warehouse. Such warehouses are traditionally divided into segments with shelves for storing dedicated items, e.g., products, components, etc. Thus, it is assumed that, from the economical perspective, implementation of a dedicated warehouse system is not justified. As a remedy, a simple KIS.ME strategy is proposed, which integrates small warehouse shelves with KIS.MANAGER with a straightforward KIS.BOX-based implementation. The proposed strategy is based on the following general principles:

Capacity: Each shelf has the desired capacity of items.

Capacity levels: Each capacity is uniformly divided into levels.

Light levels: Each capacity level is associated with a given color, e.g., level 1: green, level 2: yellow, level 3: red.

KIS.BOX implementation:

- the KIS.BOX operational LED 1 color indicates light levels;
- KIS.BOX operational LED 2 is used for presetting a given level;
- KIS.BOX Trigger detail→Current count is used for precise control and resetting of the current item count.

With the above preliminaries, it is possible to proceed to a simple yet illustrative example.

Managing an item level within the shelf

The example is implemented within Workshop 1 with KIS.BOX 1. Let us consider the item capacity levels presented in Table 3.3. There, it is evident that, when there are three or four items, then KIS.BOX 1 operational LED 1 should illuminate in green. Similarly, if there are one or two items, then KIS.BOX 1 operational LED 1 should be yellow. Finally, the zero item level is indicated with red. The above functionality can be implemented with just two rules, which are presented in Figs. 3.16 and 3.17. They can be efficiently used for decreasing the item capacity with one KIS.BOX button along with its operational LED, i.e., the KIS.BOX 1 Button 1 operational LED color. The reverse, i.e., an increase in the item capacity level, can be realized with KIS.BOX 1 Button 2. Thus, the overall performance of the system can be described by extending Table 3.3 with such a functionality, which yields Table 3.4. This means that the KIS.BOX Button 2 operational LED color indicates the capacity level which is to be reached after refilling the shelf with two items and pressing KIS.BOX Button 2. Indeed, if the shelf is full, then it is impossible to add any item, and hence the KIS.BOX Button 2 operational LED color is black, i.e., the KIS.BOX Button 2 operational LED is not lit. Two sample rules which can be used to achieve this goal are presented in Figs. 3.18 and 3.19. An obvious limitation of the above strategy is that an increase in the item level can be performed right after transferring from one capacity level to another. For example, if the capacity level is 3 and one would like to add two items, then the level should be 5, but not 4. However, all such ambiguities can be resolved by using the above-described counter reset mechanism. It should be also mentioned that the KIS.BOX initial state should be set according to Table 3.4, which corresponds to the actual item capacity level. This can be performed using the digital twin described in Sect. 2.6. The final option which can be also implemented pertains to alerting the user that the zero capacity level lasts too long, e.g., such an item capacity level lasts longer than two minutes. This can be implemented with the after x minutes optional settings, while the resulting rule is presented in Fig. 3.20.

Table 3.3 Shelf capacity management

Level	Color	Capacity
1	Green	3–4
2	Yellow	1–2
3	Red	0

The screenshot shows a rule configuration interface with three main sections: Trigger, Conditions, and Actions.
1. **Trigger:** Includes a device selector 'KIS.BOX 1', a button selector 'Button 1', and an event selector 'Pressed'. To the right, under 'Optional Settings', there is a dropdown 'After x times' set to '2'.
2. **Conditions:** Includes the same device and button selectors, followed by a 'Button 1 Color' dropdown set to 'EQUAL' with a green color swatch. A 'Flashing' checkbox is present and unchecked.
3. **Actions:** Contains two rows. The first row has 'Set LED' selected for 'Button 1 Color' with a yellow color swatch. The second row has 'Set LED' selected for 'Button 2 Color' with a green color swatch. Both rows have 'Flashing' checkboxes that are unchecked.

Fig. 3.16 Rule for changing the light level from green to yellow

The screenshot shows a rule configuration interface similar to Fig. 3.16.
1. **Trigger:** Identical to Fig. 3.16, with 'KIS.BOX 1', 'Button 1', and 'Pressed'.
2. **Conditions:** Identical to Fig. 3.16, but the 'Button 1 Color' dropdown is set to 'EQUAL' with a yellow color swatch.
3. **Actions:** The first row now has 'Set LED' selected for 'Button 1 Color' with a red color swatch. The second row remains 'Set LED' for 'Button 2 Color' with a yellow color swatch. 'Flashing' checkboxes are unchecked.

Fig. 3.17 Rule for changing the light level from yellow to red

Table 3.4 Shelf capacity management

KB Button 1 LED	KB Button 2 LED
Green	Black
Yellow	Green
Red	Yellow

Trigger ⓘ

Optional Settings

KIS.BOX 1 Button 2 Pressed -- ⓘ

Trigger Details ⓘ

Conditions ⓘ

KIS.BOX 1 Button 2 Button 2 Color EQUAL Yellow Flashing

Actions ⓘ

KIS.BOX 1 Set LED Button 1 Color Yellow Flashing

KIS.BOX 1 Set LED Button 2 Color Green Flashing

Fig. 3.18 Rule for changing the light level from yellow to green

Trigger ⓘ

Optional Settings

KIS.BOX 1 Button 2 Pressed -- ⓘ

Trigger Details ⓘ

Conditions ⓘ

KIS.BOX 1 Button 2 Button 2 Color EQUAL Green Flashing

Actions ⓘ

KIS.BOX 1 Set LED Button 1 Color Green Flashing

KIS.BOX 1 Set LED Button 2 Color Black Flashing

Fig. 3.19 Rule for changing the light level from green to black

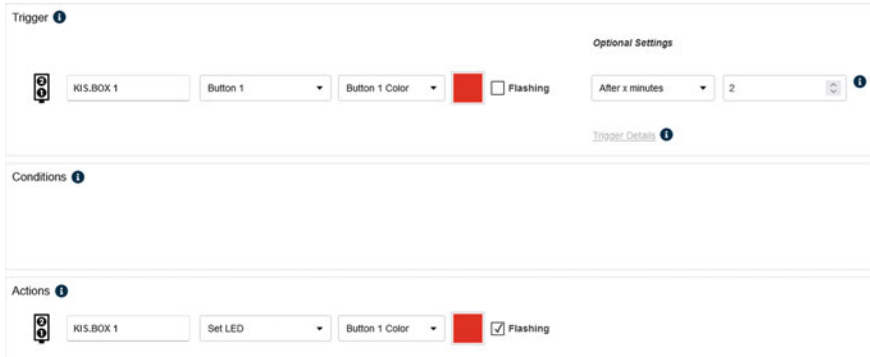


Fig. 3.20 Rule for warning about a zero item capacity with a flashing LED color

The above example can be extended to a larger number of capacity levels, which is constrained by the sum of available colors (see Table 2.2). Moreover, the information about the current status of the shelf can be further analyzed using KPIs, described in Sect. 4.1.2.

3.2 Two Points–One Transporter

The practical use case being analysed in this section pertains to a two- point transportation system equipped with one transporter only. This means that the transporter performs the desired transport actions between these two points. It is assumed that the transporter is a human-operated one, but no further details about its functionalities are stated. Transportation points are signified by two colors. In the present use case, green signifies the first point while red stands for the second one. Thus, KIS.Devices being used for a complete implementation are summarized as follows:

- KIS.BOX Green: KIS.BOX associated with the green point,
- KIS.BOX Red: KIS.BOX associated with the red point,
- KIS.BOX Transporter: KIS.BOX associated with the transporter.

The above KIS.Devices are installed in Workshop 1 along with the floorplan presented in Fig. 3.21. Additionally, the black color signifies the fact that there is no need for transportation. Under the above assumptions, the full list of transportation order events is presented in Table 3.5. This means that both KIS.BOX Green and KIS.BOX Red Button 1 operational LEDs are used to signify the transportation order according to Table 3.5. Finally, the ongoing transportation action is signified by the blue color of the respective operational LED. As a result, a decision table is proposed, which is presented in Table 3.6. Thus, the system operates with six rules, $r_1 - r_6$, and their triggers are given in Table 3.7. Implementation of rules $r_1 - r_3$ is presented in Figs. 3.22, 3.23 and 3.24. Rules $r_4 - r_6$ can be implemented analogously. Additionally, KIS.BOX Green and KIS.BOX Red Button 2 are used for canceling the transportation command. A sample implementation of this functionality for KIS.BOX

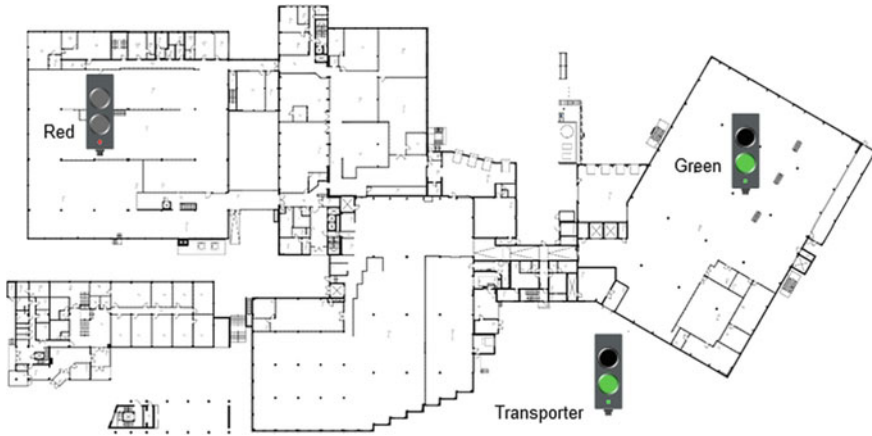


Fig. 3.21 Two points—one transporter floorplan

Table 3.5 Possible transportation scenarios

Point 1	Point 2	KB 1 Button 1 LED	KB 2 Button 1 LED
0	0	Black	Black
1	0	Green	Black
0	1	Black	Red
1	1	Green	Red

1: transport required, 0: no transportation need

Green is presented in Fig. 3.25. Finally, it should be noted that an initial condition for all KIS.BOXes is that all their operational LED colors should be black. This can be attained by using their digital twins (see Sect. 2.6). Having a complete system with an initial condition, let us consider a sample transportation action:

1. A transportation request was generated within the green point by pressing the KIS.BOX Green Button 1.
2. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs illuminate in green (see Fig. 3.21).
3. The transporter operator goes to the green point, takes the item(s) to be transported and starts the transportation action by pressing KIS.BOX Green Button 1.
4. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs illuminate in blue.
5. The transporter operator arrives at the red point, delivers the item(s) and pushes KIS.BOX Transporter Button 1.
6. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs' color is black, i.e., they do not illuminate. The transportation action is finished.

Finally, it should be noted that another transportation event from the same point can be started if the previous one has been accomplished.

Table 3.6 Decision table for the two points–one transporter system

Condition/action	r_1	r_2	r_3	r_4	r_5	r_6
KB green button 1 color is black	T	–	–	–	–	–
KB green button 1 color is green	–	T	–	–	–	–
KB transporter button 1 color is blue	–	–	T	–	–	–
KB red button 1 color is black	–	–	–	T	–	–
KB red button 1 color is red	–	–	–	–	T	–
KB transporter button 2 color is blue	–	–	–	–	–	T
KB green button 1 color is green	X	–	–	–	–	–
KB green button 1 color is blue	–	X	–	–	–	–
KB green button 1 color is black	–	–	X	–	–	–
KB transporter button 1 color is blue	–	X	–	–	–	–
KB transporter button 1 color is green	X	–	–	–	–	–
KB transporter button 1 color is black	–	–	X	–	–	–
KB red button 1 color is red	–	–	–	X	–	–
KB red button 1 color is blue	–	–	–	–	X	–
KB red button 1 color is black	–	–	–	–	–	X
KB transporter button 2 color is blue	–	–	–	–	X	–
KB transporter button 2 color is red	–	–	–	X	–	–
KB transporter button 2 color is black	–	–	–	–	–	X

3.3 Multiple Points–One Transporter

The objective of this section is to provide practical guidelines for implementing and optimizing multiple points–one transporter logistic systems using KIS.ME. Such a system is usually called a milk run one [2, 3] and comes from the dairy industry. In particular, it covers a network in which all item supply/delivery requirements of several transportation points are covered by one transporter, which visits all of them.

Table 3.7 Triggers for rules $r_1 - r_6$

Condition/action	r_1	r_2	r_3	r_4	r_5	r_6
KB green button 1 is pressed	X	X	–	–	–	–
KB transporter button 1 is pressed	–	–	X	–	–	–
KB transporter button 2 is pressed	–	–	–	–	–	X
KB red button 1 is pressed	–	–	–	X	X	–

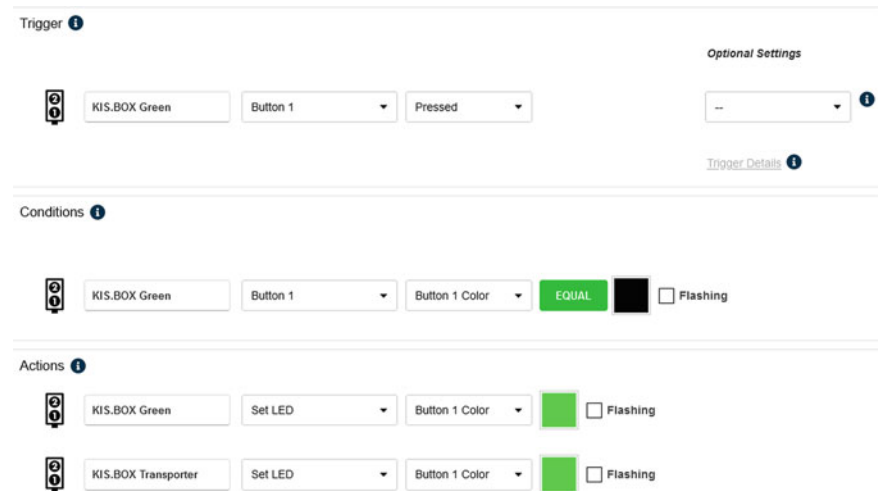


Fig. 3.22 Rule r_1 for the two points–one transporter system

Such a transporter operates within a predefined schedule, which is suitably spread over the shift time. A crucial assumption behind the economical justification of such a kind of systems is that each single point item load is smaller than the maximum transporter load volume. Consequently, the milk run strategy is frequently employed in internal plant logistics to transport, among others, items like

- raw materials,
- finished products,
- waste products.

According to the just-in-time (JIT) strategy [4, 5], the number of manufacturers using the so-called supermarket concept is proliferating. Such supermarkets can be perceived as decentralised storage areas spread over a floorplan. As a consequence, they serve as storage places for parts required by the nearby assembly lines. Using such a nomenclature, transportation points can be divided into

- assembly line stations;
- supermarket points.

Trigger 1

Optional Settings

KIS.BOX Green Button 1 Pressed

Trigger Details 1

Conditions 1

KIS.BOX Green Button 1 Button 1 Color EQUAL Flashing

Actions 1

KIS.BOX Green Set LED Button 1 Color Flashing

KIS.BOX Transporter Set LED Button 1 Color Flashing

Fig. 3.23 Rule r_2 for the two points–one transporter system

Trigger 1

Optional Settings

KIS.BOX Transporter Button 1 Pressed

Trigger Details 1

Conditions 1

KIS.BOX Transporter Button 1 Button 1 Color EQUAL Flashing

Actions 1


KIS.BOX Transporter Set LED Button 1 Color Flashing

KIS.BOX Green Set LED Button 1 Color Flashing

Fig. 3.24 Rule r_3 for the two points–one transporter system

Thus, transporter operators deliver the desired parts, which are stored in designated containers, to the assembly stations. Additionally, they collect empty containers from the assembly stations and bring them back to the supermarket points. Typically, such a process is realized with a fixed schedule and routes assigned to a given transporter, which serves a selected part of the entire assembly system, i.e., a subset of assembly stations. Such a process is accomplished by the return of the transporter to the supermarket and preparation for the next milk run tour, i.e., refiling with new

Trigger ⓘ




KIS.BOX Green

Button 2 ▼

Pressed ▼

Conditions ⓘ

Actions ⓘ




KIS.BOX Green

Set LED ▼

Button 1 Color ▼

 Flashing



KIS.BOX Transporter

Set LED ▼

Button 1 Color ▼

 Flashing

Fig. 3.25 Rule for canceling the transportation command for the green point

containers, etc. Thus, it is obvious that decentralized supermarkets, with storage points spread over the floorplan, can provide more frequent and smaller packages of parts. As a consequence, the inventory at assembly lines can be reduced, along with the elimination of relatively long-travel deliveries from one central store. Irrespective of the transportation strategy used, there are two unwanted situations which have to be prevented by appropriate part delivery [6]:

- Material shortage: This causes assembly stops resulting in an idle time;
- Enlarged safety stocks: This causes space reduction around the assembly point as well as increased inventory costs.

Furthermore, the container of parts may have an associated Kanban [7, 8] card including all important information about them. These cards are used by the Kanban system to provide permanent replacement of the consumed parts. In particular, the inventory level of each part at every assembly station is associated with the so-called Kanban number k . Thus, appropriate selection of k is crucial for preventing the above listed unappealing situations, i.e., either material shortages or enlarged safety stocks. There are several approaches which can be used to settle determination of k . However, all of them involve some heuristics along with a kind of conservativeness. As a representative example, let us recall the celebrated Toyota formula [9]:

$$k \geq \frac{C_r T_r (1 + S_f)}{K}, \quad (3.1)$$

where

- C_r is the part consumption rate per unit time, which is a function of assembly line efficiency;
- T_r is the replenishment lead time, which is a function dependent on handling transporters as well as the delivery route being used;
- K is the container capacity, which depends on the part shape/weight/volume, etc.;
- S_f is a safety factor expressing safety stock needs.

Irrespective of the replenishment strategy being used, the part consumption rate C_r , as well as the replenishment lead time T_r , is a time-varying parameter. Thus, the only way to achieve an appropriate replenishment balance is to monitor these parameters constantly.

The objective of the subsequent part of this section is to provide practical guidelines for implementing such strategies using KIS.ME. For the sake of simplicity, the proposed solution involves one transporter only. However, such conservativeness is to be eliminated in the subsequent section. As has already been mentioned, it is assumed that each transportation action is started in the supermarket points and aims at collecting the desired containers and transporting them to the assembly line stations. Sample supermarkets, located at RAFI GmbH & Co. KG, are presented in Figs. 3.26 and 3.27. As can be observed, they differ in their size as well as the transporters being used. Finally, the process is accomplished by collecting empty containers and bringing them back to the supermarket points.

Let us start with providing tools for measuring the container consumption rate. It can be given in containers per time unit. One can also imagine a situation in which it is expressed with a unified set of various container per time unit, which is required to perform a desired assembly process. The proposed strategy is based on the following general principles:

Maximum assembly point capacity: Each assembly station has a maximum level of containers which can be stored for further processing.

Capacity levels: The maximum capacity is uniformly divided into levels.

Light levels: Each capacity level is associated with a given color, e.g., level 1: green, level 2: yellow, level 3: red.

KIS.BOX implementation:

- the KIS.BOX operational LED 1 color indicates light levels;
- the KIS.BOX operational LED 2 is used for presetting a given level;
- KIS.BOX Trigger detail→Current count is used for precise control and resetting of the current container count.

A sample KIS.BOX installed at the assembly system is presented in Fig. 3.28. Note that the strategy is similar to the one used for managing a small warehouse presented in Sect. 3.1.1. Thus, it will be revisited for the purpose of a simple yet illustrative example presented in the sequel.



Fig. 3.26 A sample large supermarket at RAFI GmbH & Co. KG



Fig. 3.27 Sample small supermarket at RAFI GmbH & Co. KG

Fig. 3.28 Sample KIS.BOX installed at the assembly station



The KIS.BOX installed within the transporter is used to provide information about the selected transportation route being chosen. It is assumed that each route has an associated color, which is before the transportation action by the transporter operator. Thus, the general principles are as follows:

Route colors: Each route has an associate color, e.g., route 1: blue, route 2: turquoise, etc.

KIS.BOX implementation:

- the KIS.BOX operational LED 1 color indicates the route;
- KIS.BOX Button 1 is used to select the desired route, i.e., its color;
- KIS.BOX Button 2 is used to start/stop the transportation action.

Container usage and replenishment

Let us consider the floorplan presented in Fig. 3.29, which contains

- two supermarket points (Supermarket 1 and Supermarket 2),
- three assembly points (Assembly 1, Assembly 2 and Assembly 3),
- one transporter.

Both the assembly stations and the transporter are equipped with KIS.BOXes, i.e., KIS.BOX Assembly 1, KIS.BOX Assembly 2, KIS.BOX Assembly 3, KIS.BOX Transporter. Let us consider a sample assembly station, i.e., Assembly station 1, equipped with KIS.BOX Assembly 1, along with the container capacity levels presented in Table 3.8. From the table, it is evident that, when there are three or four containers, then KIS.BOX Assembly 1 operational LED 1 should illuminate in green. Similarly, if there is one or two, then KIS.BOX Assembly 1 operational LED 1 should be yellow. Finally, the zero container level is indicated with red. The above functionality can be implemented with just two rules. These are identical to those presented in Figs. 3.16 and 3.17, and hence they are omitted.

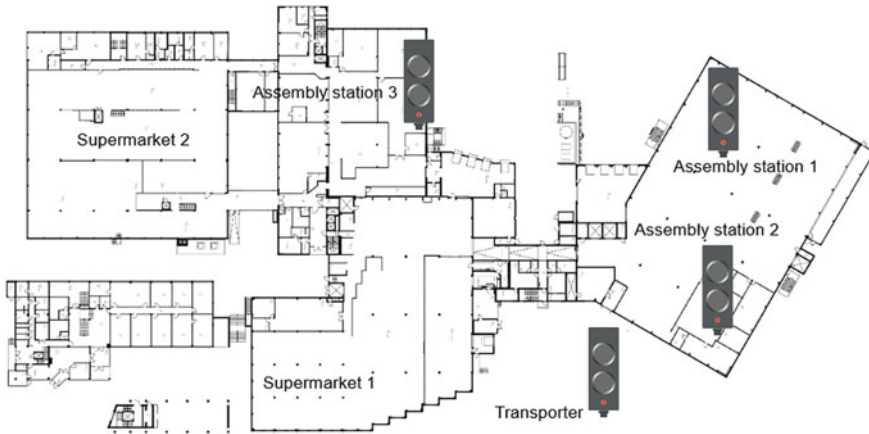


Fig. 3.29 Floorplan for the one transporter–multiple points system

Table 3.8 Assembly station container capacity levels

Level	Color	Capacity
1	Green	3–4
2	Yellow	1–2
3	Red	0

The reverse, i.e., an increase in the container capacity level, can be realized with KIS.BOX Assembly 1 Button 2. Thus, the overall performance of the system can be described by extending Table 3.8 with such a functionality, which yields Table 3.9. This means that the KIS.BOX Button 2 operational LED color indicates the capacity level which is to be reached after refilling the storage place with two containers and pressing KIS.BOX Button 2. Indeed, if the storage place is full, then it is impossible to add any container, and hence the KIS.BOX Button 2 operational LED color is black, i.e., the KIS.BOX Button 2 operational LED is not lit. The above functionality can be also implemented with two rules, which are similar to those presented in Fig. 3.18. Note also that the remaining system parameters are set in the same fashion as those presented in Sect. 3.1.1.

Such an implementation should be performed for the remaining assembly stations. Once it is completed, it is possible to analyse each assembly station’s efficiency with respect to container usage per time unit. This is, however, the objective of Sect. 3.5.

Now, let us proceed to details concerning the relations between supermarkets and assembly points. This pertains to the use of containers (sets of containers) from a

Table 3.9 Container capacity management

KB Button 1 LED	KB Button 2 LED
Green	Black
Yellow	Green
Red	Yellow

Table 3.10 Relation between supermarket points and assembly stations

Assembly station	Supermarket point 1	Supermarket point 2
1	x	–
2	x	x
3	–	x

x signifies a functional relation

Table 3.11 Set of possible routes

Route	Route color	Points
1	Blue	S2→A3→S1→A2→A1→S1→S2
2	Turquoise	S1→A1→S1
3	Green	S2→S1→A2→S1→S2
4	Magenta	S2→A3→S2
5	Red	S2→A3→S1→A2→S1→S2
6	Black	Idle state

S: supermarket point, A: assembly point

given supermarket point by a given assembly station. Such a relation is detailed in Table 3.10. This, for example, means that the final product development in Assembly station 2 requires a set of two containers, i.e., one from Supermarket point 1 and one from Supermarket point 2. This clearly means that Table 3.10 determines a set of possible routes from a supermarket point to an assembly station and back. Indeed, there is not always a need to supply all assembly stations within a given transportation cycle. Of course, there are plenty of possible replenishment strategies. For example, one can start it when the associated KIS.BOX operational LED is yellow (cf. Table 3.8), which corresponds to having one to two containers (sets of various containers) at a given assembly station. Irrespective of the criterion being used, a set of feasible routes can be determined as the one presented in Table 3.11. This set can also be visualized in the floorplan, which is presented in Fig. 3.30. According to Table 3.11, Route 1 covers all available supermarkets and assembly points, while Routes 2–5 can be perceived as its sub-routes. Thus, using them simply translates to smaller overall transportation costs and greater availability of the transporter. Note that the above set of routes is incomplete. However, it is sufficient for the illustration purposes. Finally, the KIS.BOX transporter obeys the following rules:

Route selection: The operator is pressing KIS.BOX Transporter Button 1 to select an appropriate route color. This is can be realized using a state-space model described in Sect. 2.10 with the states given in Table 3.11.

Transportation start/stop: After selecting an appropriate route the operator pushes KIS.BOX Transporter Button 2 to indicate the start of the transportation action. Once this action is accomplished, the operator pushes the button once again and the KIS.BOX operational LED 1 color turns black (idle state, cf. Table 3.11).

The implementation of the above functionality can be performed using similar mechanisms as those presented in the preceding sections, and hence it is not omitted here.

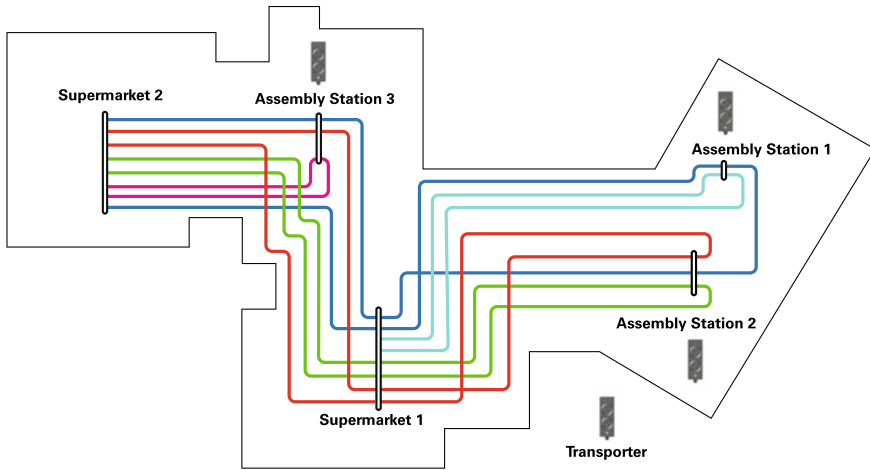


Fig. 3.30 Floorplan with a set of routes

> Transportation decision making

The transportation decisions are made based on container usage or the consumption rate, which can be directly observed in KIS.MANAGER. Thus, there are a few options which can be used to perform appropriate decisions:

Transporter operator decision: The transporter operator is equipped with a mobile device, which enables displaying the floorplan along with the current container status of each assembly station. Using such knowledge the operator makes an appropriate decision on his/her own.

KIS.MANAGER operator decision: Based on the current container status of each assembly station, the KIS.MANAGER operator sends information to the transporter operator concerning a subsequent transportation action. For example, this can be realized

- by sending an e-mail notification to the transporter operator (see Sect. 2.9);
- by directly changing the KIS.BOX Transporter operational LED 1 color to the one associated with the selected route.

Dedicated rule-based decision: The rule base in Rule engine can be extended to automatically make the transportation decision. However, such an approach requires a relatively large number of rules, and hence it is recommended for small transportation systems only.

External software decision: The information about the current container status is fed to the external software (e.g., manually or using GPIOs), which returns an appropriate decision, further passed on to the transporter operator.

Identification of position x : The KIS.BOX Transporter Zone Button 1 operational LED color signifies x coordinate of the current zone.

Identification of position y : The behaviour of the KIS.BOX Transporter Zone Button 2 operational LED color signifies coordinate y of the current zone.

The behaviour of both Button 1 and Button 2 is implemented using the state-space model principle as the one described in Sect. 2.10, while the triggers are associated with pressing either Button 1 or Button 2. As a result, the current zone can be easily set by the transporter operator using just two buttons of KIS.BOX Transporter Zone.

Sample multiple transporter system

In the example being considered, the entire floorplan is divided into 15 zones. Each one is identified by a suitable color, i.e., $x \in \{\text{blue, turquoise, black}\}$ and $y \in \{\text{blue, turquoise, black, green, magenta}\}$. There are two transporters, and each of them has an associated set of zones, which are covered by the black and red borders presented in Fig. 3.31. As can be observed, KIS.BOX Transporter 1 Zone indicates that the associated transporter is within the zone (blue,black). Similarly, KIS.BOX Transporter 2 Zone signifies that the second transporter is within the zone (black,green).

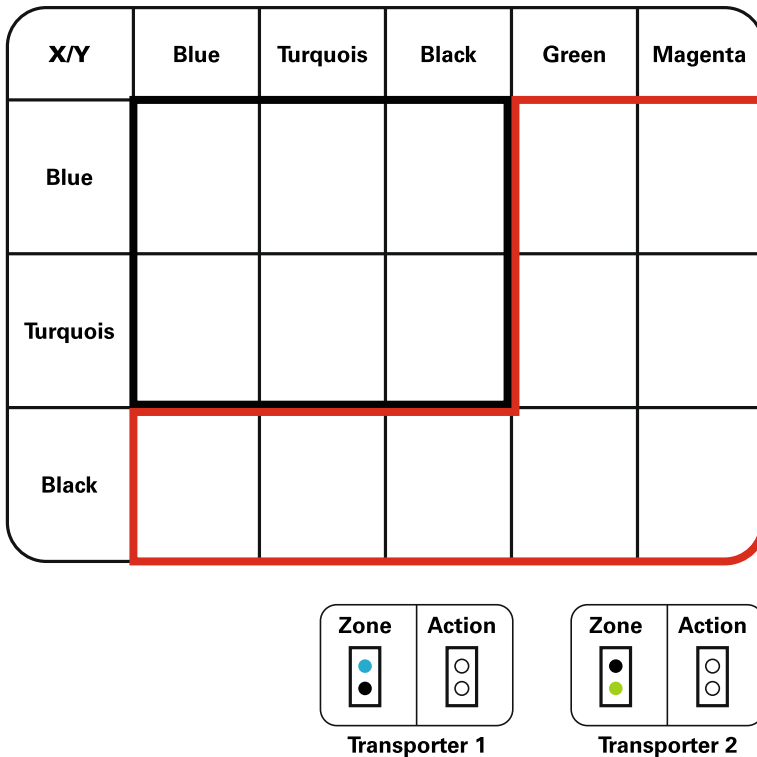


Fig. 3.31 Floorplan with zones and transporters

> **Physical zone identification**

The implementation of the proposed approach requires appropriate physical zone identification. This can be, for example, achieved by colored indicators on the borders of each zone on a given route. This will enable the transporter operator to provide appropriate modification of the current zone using KIS.BOX Transporter Zone. There are, of course, several other approaches, e.g., indoor WiFi-based position estimation, ZF Openmatics TAG Finder INDOOR [11], etc. Nevertheless, all of them inherit one common drawback, i.e., they require a separate software platform and cannot be integrated with KIS.MANAGER in a straightforward way.

Having sets of zones along with the associated transporters, it is possible to proceed to define positions of supermarkets and assembly points, which are depicted in Fig. 3.32. In the black zone, it can be observed that Supermarkets 1 serves Assembly points 1–3. In the red zone, it is assumed that Supermarkets 2–3 serve Assembly point 4. Finally, Assembly point 5 is served by Supermarket 3 only. With the above configuration, a set of routes for each set of zones can be provided. The resulting set of routes is given in Tables 3.13 and 3.14 for the black and the red set of zones, respectively. The resulting routes are portrayed in Fig. 3.33.

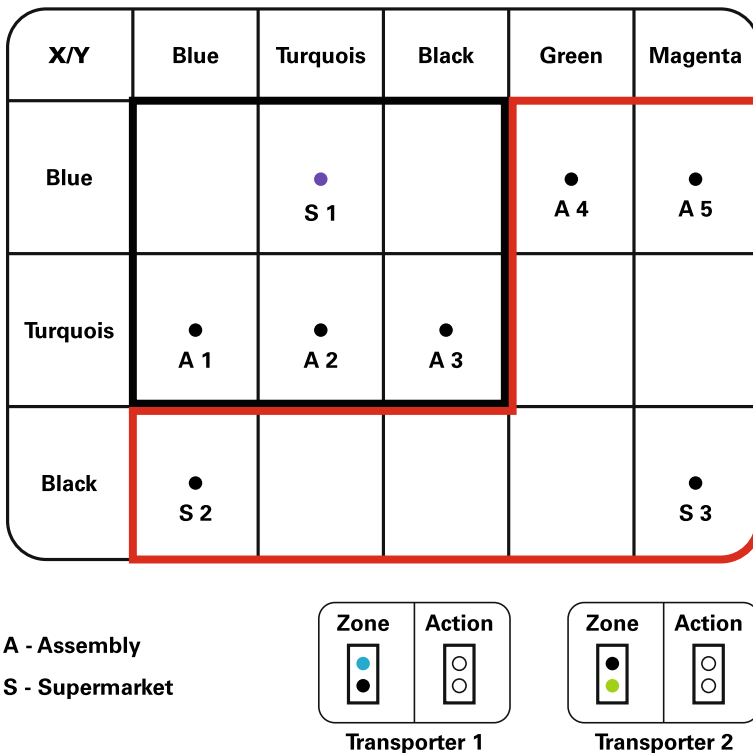


Fig. 3.32 Floorplan with zones, transporters and supermarket/assembly points

Table 3.13 Set of routes for black zones

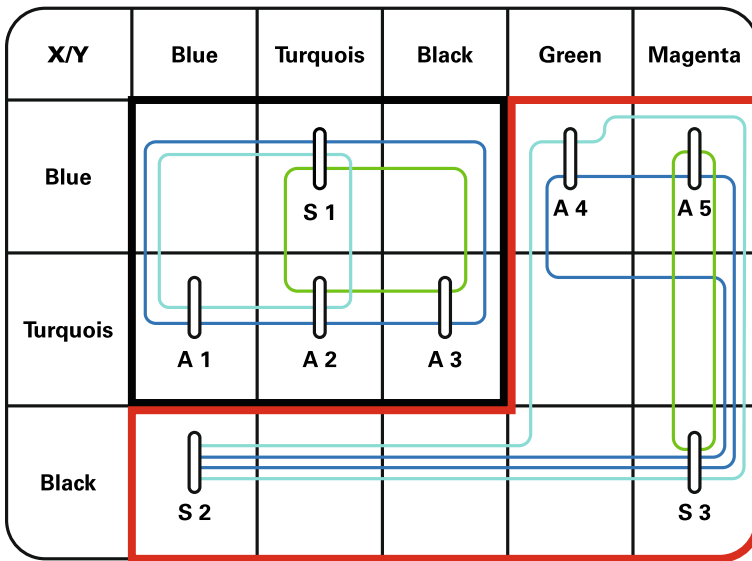
Route	Route color	Points
1	Blue	S1→A1→A2→A3→S1
2	Turquoise	S1→A1→A2→S1
3	Green	S1→A3→A2→S1
4	Black	Idle state

S: supermarket point, A: assembly point

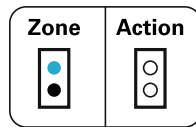
Table 3.14 Set of routes for red zones

Route	Route color	Points
1	Blue	S2→S3→A4→A5→S3→S2
2	Turquoise	S2→S3→A4→S3→S2
3	Green	S3→A5→S3
4	Black	Idle state

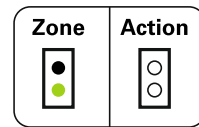
S: supermarket point, A: assembly point



A - Assembly
S - Supermarket



Transporter 1



Transporter 2

Fig. 3.33 Floorplan with feasible routes

Finally, it should be pointed out that the transportation decisions can be made in a similar way as those explained in Sect. 3.3. However, in this case, the position of each transporter is known, which may facilitate optimal transportation decisions.

3.5 Visualizing the Performance of Logistic Applications

Let us reconsider the two points—one transporter system detailed in Sect. 3.2. The entire system operates within the floorplan presented in Fig. 3.21. The objective of this section is to provide straightforward measures for calculating the performance of such a system. For that purpose, let us assume that all KIS.BOXes, i.e., KIS.BOX Green, KIS.BOX Red and KIS.BOX Transporter, are initialized in such a way that all operational LEDs are black, i.e., they do not illuminate. Under such an initial condition, reconsider the following transportation scenario:

1. A transportation request is generated within the green point by pressing KIS.BOX Green Button 1.
2. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs illuminate in green (see Fig. 3.21).
3. The transporter operator goes to the green point, takes the item(s) to be transported and starts the transportation action by pressing KIS.BOX Green Button 1.
4. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs illuminate in blue.
5. The transporter operator arrives at the red point, delivers the item(s) and pushes KIS.BOX Transporter Button 1.
6. The KIS.BOX Green and KIS.BOX Transportation Button 1 operational LEDs' color is black, i.e., they do not illuminate. The transportation action is accomplished.

The objective is to visualize and analyze

- (A) The time between the transportation request and the start of transportation action,
- (B) The time duration of the transportation action.

After analysing the above six-point transportation scenario, it can be concluded that

- the answer to point (A) can be obtained by measuring the time period for which the KIS.BOX Green Button 1 operational LED remains green (numerical value 3, see Table 2.2);
- similarly, the answer to point (B) can be obtained by measuring the time for which the KIS.BOX Green Button 1 operational LED remains blue (numerical value 0), i.e., the moment until it goes back to black (numerical value 2).

The above tasks can be conveniently realised using the Datapoint Chart, which can be configured according to the approach presented in Sect. 2.12. Let us start with introducing such a chart in Workspace 1. Such a configuration can be easily performed

Change Settings

Headline (?) Interval

Show time range selection

01 +

Asset Datapoint

Display				
	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Show Axis

Scale Axis

Display Name

Fig. 3.34 Datapoint chart for the transportation system

using KIS.BOX Green `button1ColorKpiDuration`. This process is shown in Fig. 3.34. With the chart, one can easily monitor each time measurement (A or B) visually, as shown in Fig. 3.35. A precise quantitative analysis can also be performed by downloading a CSV file using the Datapoint Chart. The content of the resulting file is presented in Fig. 3.36. The file can be further processed in a number of external computational packages, e.g., Matlab, Excel, etc. However, such data processing is also possible in KIS.MANGER, but this is beyond the scope of this chapter. Indeed, further details are provided in Chap. 4, where the concepts of KPIs are introduced (see Sect. 4.1.2). Nevertheless, one can easily extract three states from Figs. 3.35 and 3.36, while the time duration between them constitutes an answer to the above stated questions (A) and (B).



Fig. 3.35 Visualization of transportation performance

Fig. 3.36 Content of the CVS file

```
TIMESTAMP;button1ColorKpiDuration
2021-09-16T12:36:05.519Z;2
2021-09-16T12:35:20.040Z;0
2021-09-16T12:34:34.579Z;3
2021-09-16T12:33:49.100Z;2
2021-09-16T12:32:52.664Z;0
2021-09-16T12:32:05.321Z;3
2021-09-16T12:31:26.510Z;2
2021-09-16T12:30:08.533Z;0
2021-09-16T12:30:05.047Z;3
2021-09-16T12:29:30.229Z;3
```

3.6 Training Exercises

3.1 Mounting and managing an external sensor

Exercise requirements: The exercise requires access to one KIS.BOX and a sensor, e.g., a photoelectric one, compatible with an M12 connector and a 24 V power supply.

1. Using a sensor at your disposal, implement the scheme presented in Fig. 3.1.
2. Depending on the sensor type, establish its potential application related to some event counting, e.g., light on/off for a photoelectric sens.
3. Based on the application selected, prepare a suitable floorplan and place the KIS.BOX digital twin within.
4. Use the Datapoint Chart (cf. Sect. 2.12) to visualize the system and observe its current state.

3.2 Access control without RFID

Exercise requirements: The exercise requires access to one KIS.BOX and one KIS.LIGHT.

1. Prepare a set of rules which will repeatedly change the color of the KIS.BOX Button 1 operational LED according to the state sequence (cf. Sect. 2.10) presented in Table 3.15. Using the KIS.BOX digital twin, set the initial state to Stat 1.

Table 3.15 Color transition

State	KB Button LED color
1	Blue
2	Turquoise
3	Green
4	Black

2. Repeat the previous point for the KIS.BOX Button 2 operational LED;
3. Select an access-granting color configuration, e.g., KIS.BOX Button 1 operational LED is green and KIS.BOX Button 2 operational LED is blue;
4. Using the KIS.LIGHT digital twin, set its operational LED color to red, which will signify that the access is denied.
5. Implement a rule which will change the KIS.LIGHT operational LED color to green if an access-granting KIS.BOX Button color is set.
6. In the present exercise, the number of possible variations with repetitions equals

$$k^n = 2^4 = 16, \quad (3.2)$$

where n is the number of states (see Table 3.15) and k is the number of KIS.BOX buttons. Thus, using all possible eight colors (cf. Table 2.2), it is possible to have 256 configurations. Using the rule Optional settings, implement your own access control mechanism with an additional Trigger counter setting.

3.3 Hotel floor warehouse

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Let us consider a small warehouse located at a hotel floor. The warehouse serves towels for ten rooms located at that floor.
2. Assuming that all rooms are fully occupied each day and each room requires two towels a day, implement a triple-level (cf. Table 3.3) towel control system like the one presented in Sect. 3.1.1. This means that
 - Red corresponds to the zero towel level;
 - Yellow signifies the towel level which is sufficient for one day;
 - Green stands for the towel level for two days;
3. Datapoint Chart and the KIS.BOX digital twin (cf. Sects. 2.6 and 2.12) to visualize the system and observe its current state.

3.4 Hotel two-floor warehouse

Exercise requirements: The exercise requires access to two KIS.BOXes, e.g., KIS.BOX 1 and KIS.BOX 2.

1. Having a KIS.BOX 1 with an associated single floor towel warehouse implementation, perform an identical implementation for KIS.BOX 2. It will correspond to another floor, which obeys the same rules like the previous one.

2. Prepare a floorplan of these two floors and locate the KIS.BOX 1 and KIS.BOX 2 digital twins within it (cf. Sect. 2.6).
3. Use the Datapoint Chart (cf. Sect. 2.12) to visualize the system and observe its current state.

3.5 Small warehouse with access control

Exercise requirements: The exercise requires access to two KIS.BOXes, e.g., KIS.BOX 1 and KIS.BOX 2.

1. Extend the access control strategy developed in Exc. 3.2 by setting the KIS.BOX 1 digital output 1 to the On state when access is granted. The KIS.BOX digital output 1 should be Off when access is denied. (Note: The exercise can be extended by connecting the KIS.BOX 1 digital output 1 to real access control, e.g., an electronic lock.)
2. Using KIS.BOX 2, implement a single-shelf small warehouse management system according to the approach presented in Sect. 3.1.1 with an additional access control feature developed in the preceding point.

3.6 Two points—one transporter revisited

Exercise requirements: The exercise requires access to three KIS.BOXes, e.g., KIS.BOX Point 1, and KIS.BOX Point 2 and KIS.BOX Transporter.

1. Implement the two points—one transporter system described in Sect. 3.2.
2. Implement Datapoint Chart visualizing the performance of the above system (see Sect. 3.5).
3. Using a sheet of paper, prepare a fictitious transportation scenario with a set of feasible transport orders and transportation times.
4. Perform the above scenario using the implemented KIS.ME-based transportation monitoring system.

3.7 Digitalization of an assembly system

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Let us consider a single container that is fed to the assembly system. In particular, the container covers the following parts (digits):

$$C = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. \quad (3.3)$$

2. The assembly task boils down to writing down the subsequent digits from the container. Thus, if all letters are used, then the container is empty and another one can be processed assuming that it is available.
3. Employ a system capable of measuring assembly effectiveness, i.e., the time of writing all letters from (3.3).
4. Use the three-stage container capacity management system (see Sect. 3.3, Table 3.9).

3.8 Digitalization of an assembly system continued

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Let us consider an assembly system which is fed with a set of two containers, required for the assembly process. In particular, the first one is described by (3.3) while the second one is given by

$$C_l = [a, b, c, d, e, f, g, h, i, j]. \quad (3.4)$$

2. The assembly process boils down to writing a digit from container C and then a corresponding letter from container C_l . If all digits and letters are used, then the assembly is accomplished.
3. Use the assembly efficiency and container capacity management system developed in Exc. 3.7 for the assembly system being considered.

3.9 Monitoring transporters' current position zone

Exercise requirements: The exercise requires two KIS.BOXes, e.g., KIS.BOX Transporter 1 Zone and KIS.BOX Transporter 2 Zone.

1. Let us consider the floorplan presented in Fig. 3.31.
2. Using the approach proposed in Sect. 3.4, implement a set of rules which make it possible to indicate current position zones of Transporter 1 and Transporter 2.

3.7 Concluding Remarks

The objective of this chapter was to provide a set of tools which can be used for quick design and integration of logistic applications. In particular, the chapter started with extending KIS.Devices with RFID interfaces, which enable access control regarding the assets being monitored within KIS.MANAGER. Two kinds of access control were introduced, namely, general and individual. Within the former, the users are treated in the same way, i.e., each user having a suitable RFID tag is treated identically. Contrarily, in the latter, each user is uniquely recognized by KIS.MANAGER, which makes it possible to develop a tailored access control hierarchy. The rest of the chapter was concerned with a logistic application. It started with a way of using KIS.Devices for monitoring a small warehouse. Subsequently, a digitalization of one transporter operating between two points was introduced and analysed. In particular, a set decision table was presented, which handles all possible routes between these two points. An alternative concept was introduced for one transporter operating between multiple points. Each point was defined as an assembly station, which has a certain capacity of containers with parts. Such a capacity is required to assure an appropriate and efficient assembly flow. Thus, the task of the transporter was to provide an appropriate container flow between container supermarkets and assembly points. For that purpose, a KIS.ME-based optimization was proposed. Subsequently, the concept was extended to multiple transporters operating in specified zones. Note that each zone was uniquely identified using coordinates (x,y) translated into two colors visible on KIS.BOX, identifying the current position of a transporter. Finally, a set of training exercises was provided, which summarize the knowledge covered within this chapter.

References

1. G. Richards, *Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse* (Kogan Page Publishers, London, 2017)
2. M. Baudin, *Lean Logistics: The Nuts and Bolts of Delivering Materials and Goods* (CRC Press, New York, 2005)
3. H.S. Kilic, M.B. Durmusoglu, M. Baskak, Classification and modeling for in-plant milk-run distribution systems. *Int. J. Adv. Manuf. Tech.* **62**(9), 1135–1146 (2012)
4. S. Emde, N. Boysen, Optimally locating in-house logistics areas to facilitate JIT-supply of mixed-model assembly lines. *Int. J. Prod. Econom.* **135**(1), 393–402 (2012)
5. S. Emde, N. Boysen, Optimally routing and scheduling tow trains for jit-supply of mixed-model assembly lines. *Eur. J. Operat. Res.* **217**(2), 287–299 (2012)
6. M. Faccio, M. Gamberi, A. Persona, Kanban number optimisation in a supermarket warehouse feeding a mixed-model assembly system. *Int. J. Prod. Res.* **51**(10), 2997–3017 (2013)
7. Ch. Huang, A. Kusiak, Overview of Kanban systems. *Int. J. Comput. Int. Manuf.* **9**(3), 169–189 (1996)
8. C.S. Kumar, R. Panneerselvam, Literature review of JIT-KANBAN system. *Int. J. Adv. Manuf. Technol.* **32**(3–4), 393–408 (2007)
9. Y. Sugimori, K. Kusunoki, F. Cho, S. Uchikawa, Toyota production system and Kanban system materialization of just-in-time and respect-for-human system. *Int. J. Prod. Res.* **15**(6), 553–564 (1977)
10. M. Witczak, P. Majdzik, R. Stetter, B. Lipiec, A fault-tolerant control strategy for multiple automated guided vehicles. *J. Manuf. Syst.* **55**(4), 56–68 (2020)
11. Zf openmatics. https://www.zf.com/products/en/connectivity/products_52107.html. Accessed: 14 Sept 2021

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 4

Implementing and Using Essential Statistical Process Control



4.1 Data Processing Definitions

The objective of the preceding chapters was to introduce essential features of KIS.ME (cf. Chap. 2) and provide a list of typical logistic use cases for which it can be employed (cf. Chap. 3). Irrespective of the kind and structure of the system being developed, it involves a number of KIS.Devices, which provide several Datapoints (see Sect. 2.7). To make this chapter self-contained, let us recall that a Datapoint is a read only variable, which corresponds to a possibly time-varying property of a KIS.Device. It can be also defined as an exchanged value between the KIS.Device and KIS.MANAGER. In the preceding chapters, Datapoints were employed for developing the rules governing system behaviour as well as to graphically visualize it using Datapoint Charts. In this chapter, it will be shown how to process and analyse Datapoints. Let us recall that they are updated based on events associated with their current status. This means that, if the status of KIS.Devices changes, then an appropriate update message is sent to KIS.MANAGER. Having access to Datapoints, one can process them further by using scripts prepared using the FLEX programming language (see Appendix A). Such a processing procedure can be performed either instantly (real time) or over a predefined processing period which can be set to either 15, 30 or 60 min. For that purpose, let us introduce two definitions:

Calculated Datapoints (CDPs): FLEX language-based scripts enabling instant processing of Datapoints,

Key performance indicators (KPIs): FLEX language-based scripts enabling processing of Datapoints over a predefined processing period.

The concept of CDPs is rather straightforward, and as a simple example one can imagine a CDP script transforming frequency of KIS.BOX digital input from mHz to an equivalent period, i.e., $T = \frac{1}{f}$, where T stands for the period and f is the frequency. Contrarily, the concept of KPIs requires further explanation. For that purpose, let us consider principles for the calculation and analysis of KPIs, which are

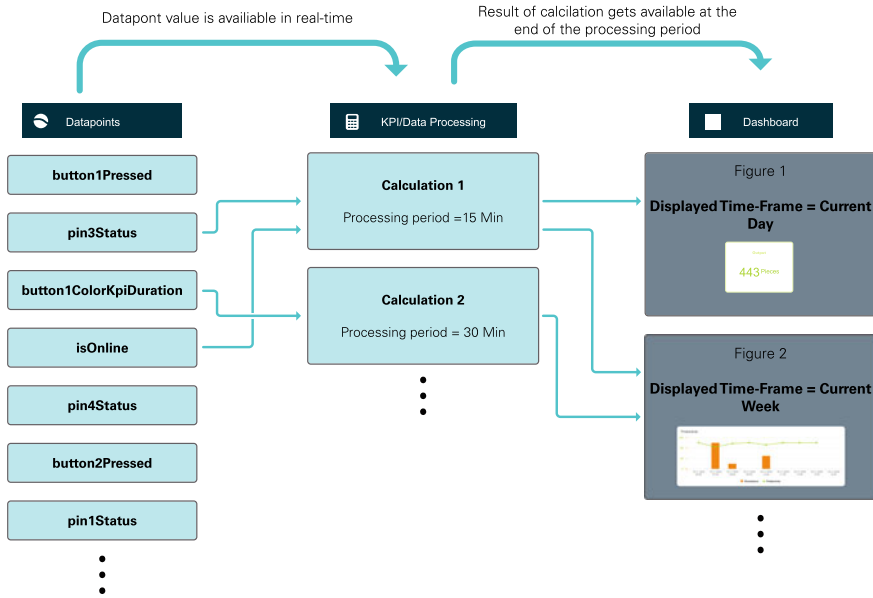


Fig. 4.1 Principles for the calculation and analysis of KPIs

presented in Fig. 4.1. It is possible to define a number of scripts performing a number of calculations based on selected Datapoints (cf. Calculation 1 and Calculation 2 in Fig. 4.1). Such KPI scripts can, of course, be defined over different processing periods while the results of the calculations are available at the end of such periods. Finally, they can be displayed and analysed using various widgets (see Sect. 2.5).

KPI: An introductory aggregation example

Let us consider sample KPIs, which provide some calculation results every 15 min. Such a process is illustrated in Fig. 4.2. The results provided can be further aggregated using various strategies. As can be observed, each KPI is calculated using 15-min processing period. From the start of the process one can observe that there are four such 15 min processing time windows. This clearly means that sample aggregations of the sum and average values of the calculation being considered can be summarized in Table 4.1.

The aggregation mechanism is to be clearly illustrated and explained in the subsequent sections of this chapter.

Furthermore, FLEX language commands available in KIS.ME are divided into five groups, which are given in Table 4.2. As a final conclusion, it can be stated that the Aggregations and Intervals commands are limited to KPIs only (see Appendix A for a detailed explanation).

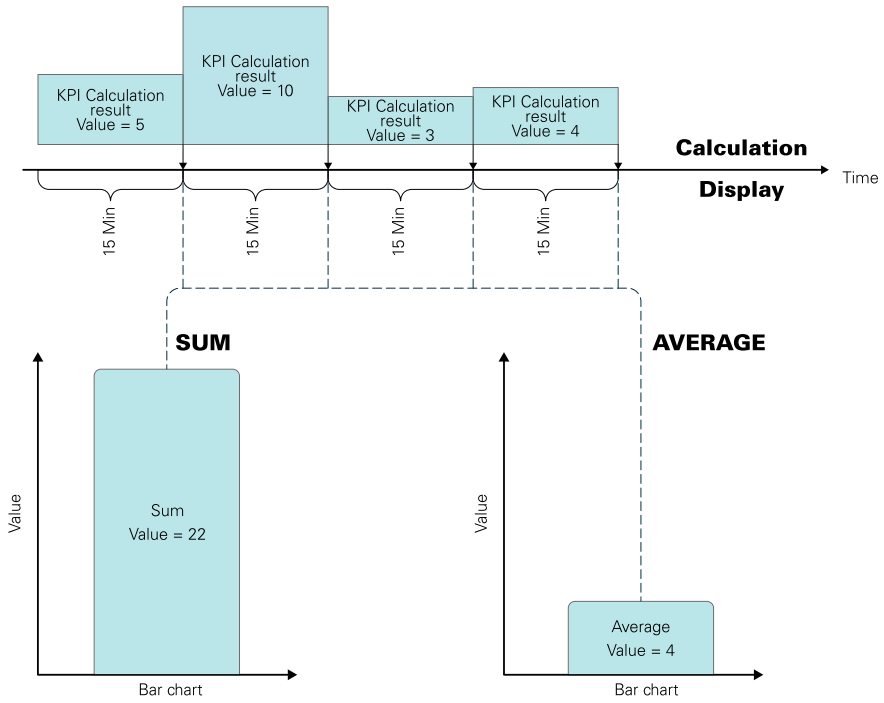


Fig. 4.2 Sample KPI aggregation calculations

Table 4.1 Evolution of a sample KPI as well as its sum and average aggregations

Processing time window	KPI calculation result	Sum	Average
1	5	5	5
2	10	$5 + 10 = 15$	$\frac{5+10}{2} = 7.5$
3	3	$5 + 10 + 3 = 18$	$\frac{5+10+3}{3} = 6$
4	4	$5 + 10 + 3 + 4 = 22$	$\frac{5+10+3+4}{4} = 5.5$

Table 4.2 KIS.ME FLEX language commands

Group name	KPI	CDPs
Aggregations	YES	NO
Intervals	YES	NO
Numeric	YES	YES
Comparison	YES	YES
Miscellaneous	YES	YES

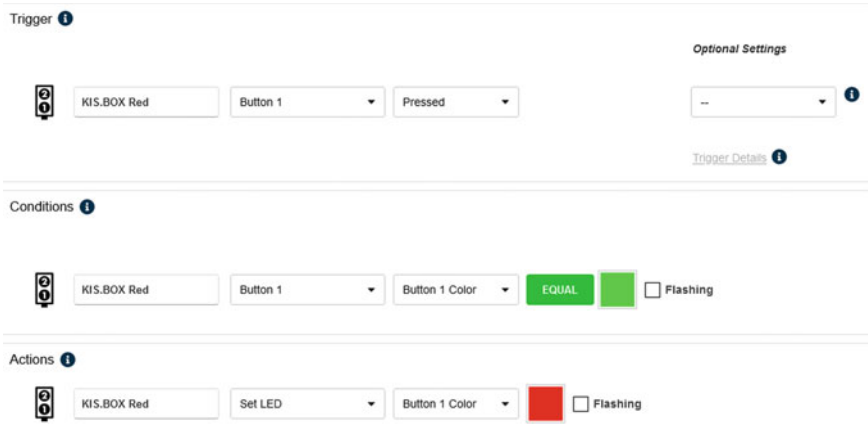


Fig. 4.3 Transition rule Green→Red for KIS.BOX Red

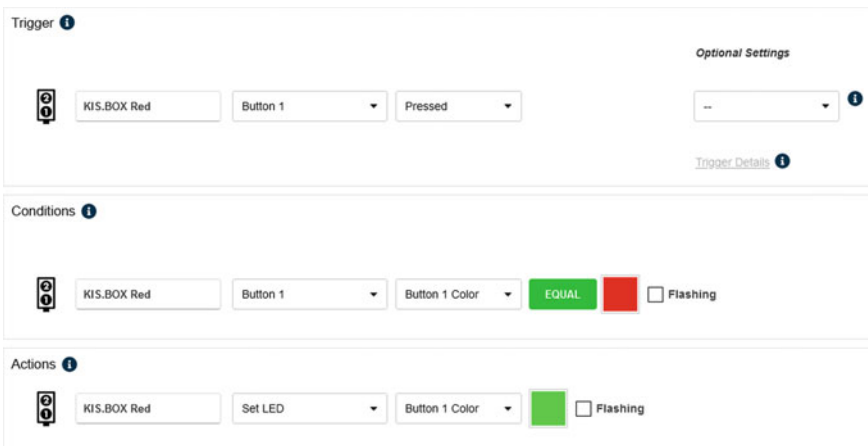


Fig. 4.4 Transition rule Red→Green for KIS.BOX Red

4.1.1 Calculated Datapoints

The objective of this section is to provide a practical example concerning the design of sample CDPs. The example is implemented in Workshop 1 and uses KIS.BOX Red. For illustrative purposes, two simple rules were implemented, which are presented in Figs. 4.3 and 4.4. These two rules transfer KIS.BOX Red from one state to another. These states correspond to either the green or the red color of the KIS.BOX Red Button 1 operational LED. The same rules were implemented for KIS.the BOX Red Button 2 operational LED, but they are omitted for brevity. In both cases, the rule trigger is associated with pressing either the first or the second KIS.BOX Red button. Finally, the initial states of are initialized using the KIS.BOX Red digital twin (cf.

Sect. 2.6), i.e., the corresponding operational LED colors are set to green. Let us start with employing a sample command, which is called Counter and belongs to the Miscellaneous group (cf. Appendix A.15). The syntax and the functionality of the above command can be summarized as follows:

$$y = \text{Counter}[x, b_p] \tag{4.1}$$

and

$$\begin{aligned} &\text{if } x_c \geq x_p \text{ then } y = x_c - x_p, \\ &\text{else } y = b_p + x_c - x_p, \end{aligned} \tag{4.2}$$

where x is a possibly varying value while x_c, x_p signify x at $c > p$ time instances, whilst b_p stands for a possibly time-varying bias. Let us proceed to defining a CDP employing such a command. For that purpose, KIS.BOX Red should be selected from the available assets (Main menu → Assets). Subsequently, one should select the KPI/Data processing button and use the Create blank option. As a result, a new Data Processing Definition can be formulated as the one presented in Fig. 4.5. The data processing definition was selected to be CDP and is called Counterdmo. Its entire implementation boils down to $y = \text{Counter}[x, 0]$, with x being an input Datapoint button1ColorKpiDuration. According to the above-described rules (cf. Figures 4.3 and 4.4), x may have two values only, i.e., 3 (green color) or 5 (red color). Thus, according to (4.2), Counterdmo can return only two values, i.e., either $y = 5 - 3 = 2$ ($x_c = 5, x_p = 3$) or $y = 0 + 3 - 5 = -2$ ($x_c = 3, x_p = 5, b_p = 0$). This means that Counterdmo simply calculates the difference between consecutive values of x . To visualize this graphically, let us include the Datapoint Chart within the KIS.BOX Red dashboard (see Sect. 2.7). The resulting plot is portrayed in Fig. 4.6.

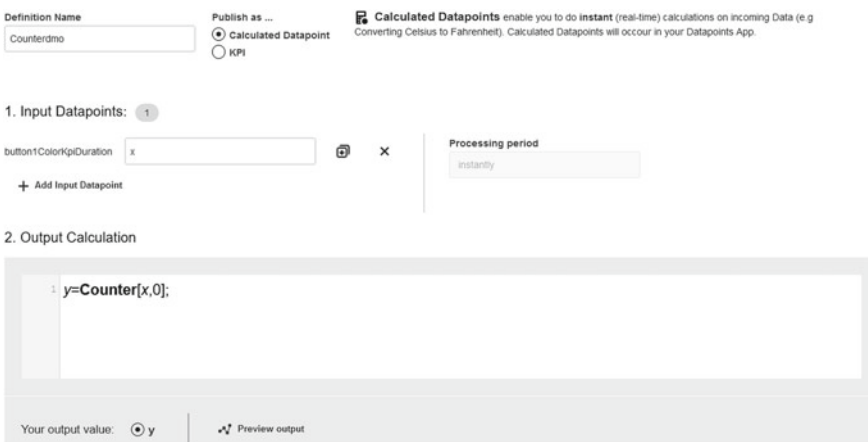


Fig. 4.5 Implementation of the Counterdmo CDP



Fig. 4.6 Variable x (red) and Counterdm0 CDP calculation (green) for $y = \text{Counter}[x, 0]$

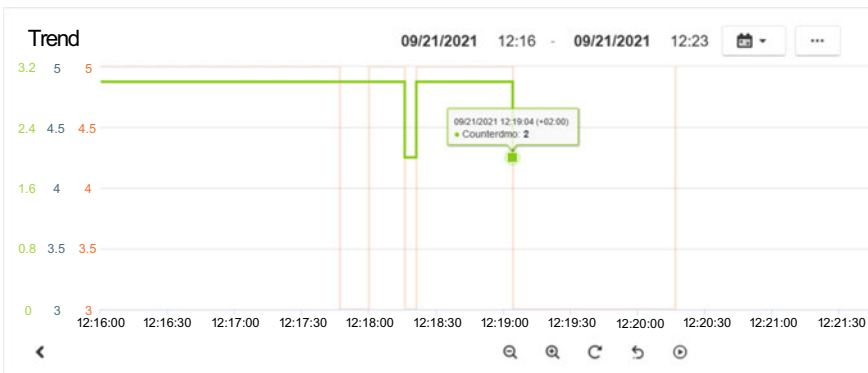


Fig. 4.7 Variable x (red) and Counterdm0 CDP calculation (green) for $y = \text{Counter}[x, x]$

Finally, it is important to underline the fact that, if x remains unchanged during a possible reconnection of the KIS.Device, then $x_p = x_c$, and hence $y = 0$ is obtained. Let us proceed to a more advanced usage of the Counter command. Counterdm0 can be easily redefined with a new syntax, i.e., $y = \text{Counter}[x, x]$. According to (4.2), Counterdm0 can return only two values, i.e., either $y = 5 - 3 = 2$ ($x_c = 5$, $x_p = 3$) or $y = 5 + 3 - 5 = 3$ ($x_c = 3$, $x_p = 5$, $b_p = 5$). These results are visualized in Fig. 4.7. Let us consider another command, which simply filters the data by permanently returning the last value satisfying a given logical formula. For that purpose, let us define another CDP, which will be called Filterdm0 and will employ the following syntax: $y = \text{Filter}[\text{Equal}[x, 3]]$ (or, equivalently, $\text{Filter}[x == 3]$). If x can be either 3 or 5, then after the first occurrence of 3 the answer of the Filterdm0 CDP remains at the level of 3 (see Fig. 4.8). It should be noted that the argument of the Filter command can be any logical relation, e.g., $x \geq 3 \&\& x \leq 5$.

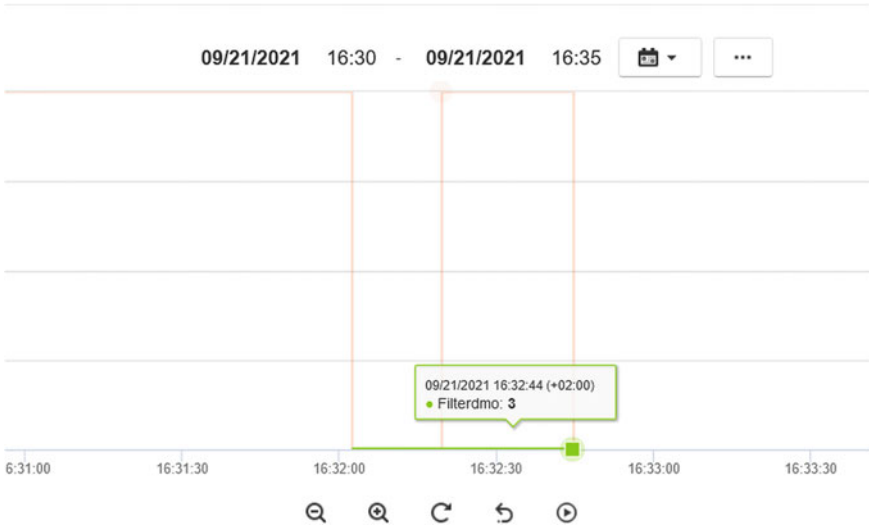


Fig. 4.8 Variable x (red) and Filterdmo CDP calculation (green) for $y = \text{Filter}[\text{Equal}[x, 3]]$

Plotting workers' idle state

Let us consider two workers performing identical works at a single assembly station. Both of them use KIS.BOX Red to indicate two states:

- Assembly in progress: exemplified by the red color of operational LEDs,
- Idle: exemplified by the green color of operational LEDs.

This means that Worker 1 uses KIS.BOX Red Button 1 while Worker 2 utilizes KIS.BOX Red Button 2. The rules for switching between assembly and idle states are given in Figs. 4.3 and 4.4. The problem is to indicate the time in which both workers are in idle state, i.e., the Button 1 and Button 2 operational LEDs are green. Let us consider a CDP performing such an action, which is called Idle and is given in Fig. 4.9. It uses input variables x and z , which are defined with two KIS.BOX Red Datapoints, i.e., `button1ColorKpiDuration` and `button2ColorKpiDuration`, respectively. The calculation involving such variables utilizes the `If[]` command (cf. Appendix A). If both x and z equal 3, then the KPI returns 1 and 0 otherwise. Note that 1 signifies the fact that both workers are in idle state, while the opposite situation means that at least one of them is performing an assembly process. As a result, Figs. 4.10 and 4.11 present the state evolution of Worker 1 and Worker 2. Finally, Fig. 4.12 clearly indicates the time periods for which both workers are in idle state. In spite of the simplicity of the CDP, it may have various practical applications, i.e., optimization of work distribution, part delivery schedules, etc.

Definition Name Publish as ... Calculated Datapoint KPI

Calculated Datapoints enable you to do **instant** (real-time) calculations on incoming Data (e.g. Converting Celsius to Fahrenheit). Calculated Datapoints will occur in your Datapoints App.

1. Input Datapoints: 2

button1ColorKpiDuration

button2ColorKpiDuration

+ Add Input Datapoint

Processing period

2. Output Calculation

```
y=If[x==3 && z==3,1,0];
```

Your output value: y

Fig. 4.9 Idle CDP



Fig. 4.10 States of Worker 1



Fig. 4.11 States of Worker 2



Fig. 4.12 Evolution of the Idle CDP

The examples presented in this section are very simple and involve one-line commands. There are, of course, no restrictions behind developing multiple-line CDPs. Moreover, own variables can be used without any prior declarations, i.e., a variable begins its lifetime after assigning to it a value, which is realized by the following program:

Sample two-line code

```
MyVar=x+3;  
OutVar=Counter[MyVar,0];
```

> CDPs versus the Datapoint range

Note also that each CDP can operate with Datapoints of one KIS.Device only. It is possible to include Datapoints from different KIS.Devices indirectly, i.e.,

- by using digital inputs of the KIS.Device for which the CDP is being developed;
- by Rule engine to transmit Datapoint values the different KIS.Devices to the one for which the CDP is being developed.

Finally, let us recall that a complete list of the FLEX commands along with representative examples is given in Appendix A.

4.1.2 Key Performance Indicators

As indicated in Figs. 4.1 and 4.2, the functional nature of KPIs is significantly different than the one of CDPs. Indeed, CDPs process incoming data directly without any

aggregation mechanisms, e.g., summation (cf. Figure 4.2). Thus, the objective of this section is to provide illustrative design examples pertaining to such aggregations as well as working on data within predefined processing periods.

Introduction to KPI design

Let us consider a worker performing an assembly process (cf. Sect. 4.1.1). KIS.BOX Red is used in this case to indicate two states:

Assembly in progress: exemplified by the red color of the KIS.BOX Red Button 1 operational LED,

Idle: exemplified by the green color of the KIS.BOX Red Button 1 operational LED.

The rules governing transitions between these two states are given in Figs. 4.3 and 4.4. Similarly as in Sect. 4.1.1, one can also easily configure a Datapoint Chart displaying the successive transitions between these two states, which simply includes the values of the `button1ColorKpiDuration` Datapoint. The objective of this introductory KPI example is to provide a periodic calculation of the number of products being assembled. Thus, under a state transition strategy being used, the KPI should periodically calculate the number of transitions of the KIS.BOX Red Button 1 operational LED from red to green. To settle the KPI implementation, the simple instruction $y = \text{Sum}[\text{If}[x == 3, 1, 0]]$ is used, where x stands for the `button1ColorKpi` Datapoint (see Appendix B). The remaining KPI parameters are as follows:

Processing period: 15 min;

Processing start: the data of starting calculations of the KPI, which is 09/22/2021;

Starting hour: the starting hour is 14.00;

Initial value: no values from previous periods are taken into account.

A new KPI design can be initiated in the same way as that of a CDP, and hence the resulting configuration window is given in Fig. 4.13, whilst the new KPI is called `Sumdmo`. It should be noted that KPIs cannot be displayed with the Datapoint Chart. This is caused by the fact that they should be aggregated using a desired operation (cf. Fig. 4.2), which can be one of the following:

- SUM,
- MIN,
- MAX,
- AVERAGE.

Such an aggregation can be achieved with suitable widgets (Sect. 2.5), which will be discussed in detail in Sect. 4.3. Following the approach described in Sect. 2.5, let us introduce the Single Value KPI widget, which will accompany the Datapoint Chart displaying the evolution of the `button1ColorKpiDuration` Datapoint. A sample configuration of the above widget is presented in Fig. 4.14. As can be observed, the aggregation type is SUM. Moreover, the date range of interest is the current

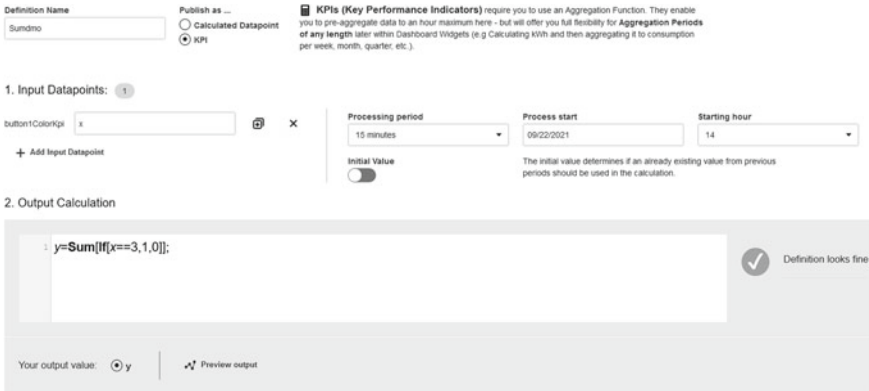


Fig. 4.13 Implementation of the Sumdmo KPI

day. However, according to Fig. 4.13, the Sumdmo KPI begins its lifetime at 14.00. Figure 4.15 presents the obtained results, which were recorded during one hour. They are summarized in Table 4.3 and can be easily calculated using the Datapoint Chart, i.e., the number of transitions from red to green (5–3). Thus, it is evident that the SUM of KPI values is equal to 10 whilst the MIN and MAX are equal to 2. As a consequence, the AVERAGE is equal to 2.5. Irrespective of the relative simplicity of the example being considered, it may have several practical applications pertaining to production/transportation flow and efficiency monitoring.

The objective of the remaining part of this section is to provide some template KPIs along with their prospective applications, which pertains to

- monitoring task realization durations,
- counting the number of button presses,
- counting the number of assembled products per time unit,
- first pass yield (FPY), i.e., the ratio between outgoing and incoming process units.

Monitoring task realization durations

Let us reconsider a worker from the previous example, which is equipped with KIS.BOX Red indicating its current working mode, i.e., either assembly or idle periods. The first objective is to implement the KPI which will calculate the sum of the assembly times within a 15-min processing period. Let us start with selecting an appropriate Datapoint, which is simply `button1ColorKpiDuration`. Thus, during the assembly process, it returns 5, which corresponds to the red color. This means that the implementation of the KPI should boil down to the following steps:

Headline
kpiSingleValue

Subtitle

KPI name **Color**

Unit

Aggregation type
SUM

Date Range
Current Day

09/22/2021 00:00 - 09/23/2021 00:00

Working Shift

Fig. 4.14 Configuration of the single value KPI widget

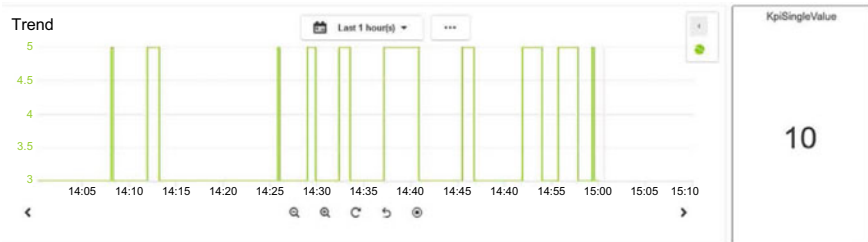


Fig. 4.15 Sumdmo KPI aggregation

1. Determine assembly states within the processing periods along with their durations.
2. Sum all the above durations.
3. The durations are expressed in milliseconds, and hence they should be converted to seconds and then rounded appropriately.

The implementation covering all the above steps is presented in Fig. 4.16, whilst a detailed explanation behind each command is given in Appendix A. Note that

Table 4.3 Evolution of the Sumdmo KPI

Processing period (h)	KPI value
14.00–14.15	2
14.15–14.30	2
14.30–14.45	2
14.45–15.00	4

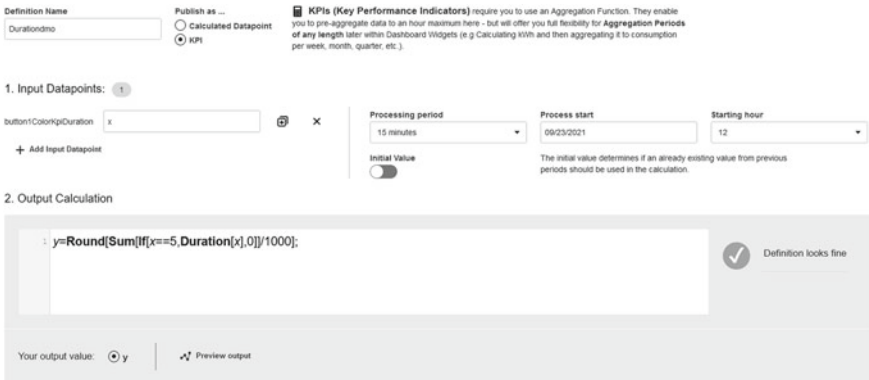


Fig. 4.16 Implementation of the Duration KPI

the calculation of the Duration KPI is started at 12.00 and no values from previous processing periods are taken into account. Let us proceed to the second objective, which pertains to calculating

- the total sum of assembly durations for all processing periods,
- the total average of assembly durations for all processing periods,
- the minimum sum of assembly durations per processing period,
- the maximum sum of assembly durations per processing period.

As previously, the above tasks can be easily implemented by installing the KPI Single Value widgets within the dashboard. Each of them should, of course, perform a different aggregation, i.e., SUM, AVERAGE, MAX, and MIN. Finally, a sample implementation along with the obtained results is presented in Fig. 4.17. Note that the maximum sum of durations per a processing period can be equal to $15 \times 60 = 900[s]$, which expresses the situation in which the assembly is permanently performed during 15 min. Contrarily, the minimum sum of durations per processing period can be equal to zero, which means that no assembly was performed within a processing period.

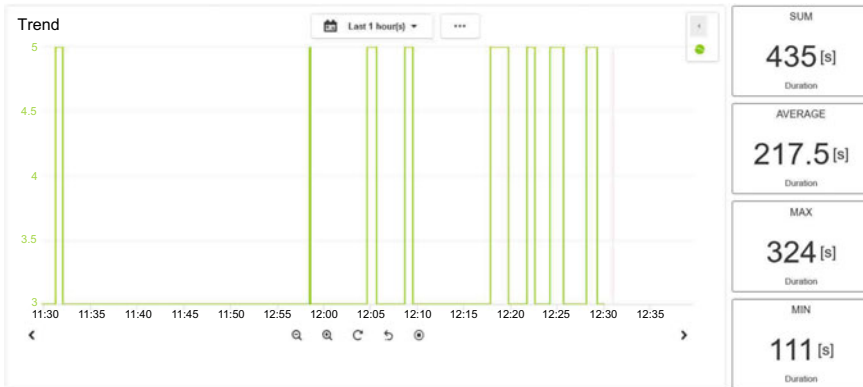


Fig. 4.17 Duration KPI aggregations

Number of assembled products per time unit

The example being considered operates within the same infrastructure as the previous one. Let us imagine that each push of a button corresponds to one product being assembled. Thus, let us start by selecting an appropriate Datapoint, which is `button1Pressed` and has an alias name x within the KPI definition. This Datapoint operates in a very specific way, i.e., it stores the timestamps corresponding to the beginning of pressing actions only. Thus, it is enough to calculate the number of logical true values of x within the processing period. Subsequently, the accumulated number of transitions has to be normalized over the time interval spent on realizing a given task. The above-stated objective can be realized with the KPI given by the following program:

Product per minute KPI

```
ProductSum = Sum[If[x,1,0]];
Productivity = Round[ProductSum / (Interval[] / 60000)];
```

The first line is responsible for calculating the number of products, i.e., the total of KIS.BOX touches. The second one normalizes `ProductSum` over the respective time interval. Finally, the achieved results are given in milliseconds, and hence they have to be further normalized. Note also that the rounding operation is introduced to get a result, which is an integer value.

First pass yield

Let us consider a conventional product quality check point. It has two digital outputs, which are directly connected with KIS.BOX Red digital inputs. They provide a false–true–false sequence when a products passes (Input 1) or fails (Input 2) the quality test. Let us start with noting that Input 1 and Input 2 are represented by two Datapoints, i.e., `input1Status` and `input2Status`, respectively. Let us define two variables which are to be associated with these Datapoints, i.e., `passed` and `failed`. Let us recall that the percentage FPY is expressed by the ratio

$$\text{FPY} = \frac{p}{p + f} \times 100, \quad (4.3)$$



where p and f stand for the number of failed and passed products, respectively. Finally, the KPI implementing (4.3) is given by the following:

First pass yield KPI

```
NumberPassed = RisingEdge[passed];
NumberFailed = RisingEdge[failed];
Total = NumberPassed + NumberFailed;
FPY = If[Total>0, Round[NumberPassed/Total*100], 0];
```

As in the previous example, the first two lines are responsible for calculating the p and f underlying (4.3). Subsequently, the total sum of products being tested is obtained. Finally, FPY is calculated according to (4.3) along with suitable rounding.

> Sharing CDPs and KPIs

Using Main menu→Assets, and then selecting the desired KIS.Device, one can choose KPI/Data processing . As a result, a full list of KPIs and CDPs defined for this KIS.Device is displayed. Moreover, each of them can be shared with compatible KIS.Devices of a designed asset group. For that purpose, the Share option should be used and the desired asset group chosen, e.g., Workspace 1. Once the sharing procedure is performed, the selected CDPs/KPIs are visible after using KPI/Data processing  within a designated asset group. However, sharing should be interpreted in such a way that a given CDP/KPI is inherited for all assets in a designated asset group. Indeed, Datapoints used within the shared CDP/KPI are taken from the inheriting KIS.Device instead of the original one.

4.2 Statistical Measures: Location and Variability

The objective of this section is to introduce essential concepts of statistical process control (SPC). However, before going into details, it is necessary to define a process [1–3] as follows:

> Process

A process is everything what is needed for transforming an input into an output for a customer.

Thus, SPC can be perceived as a quality control method, which employs statistical approaches to observe and control a process. This may result in keeping the desired process efficiency, assembling the demanded number of products with a desired quality, etc. This means that SPC deals with understanding and managing variability associated with a given process. For that purpose, a process parameter, i.e., a quantitative variable, which reflects the process quality must be selected. Irrespective of the process parameter being observed and controlled, it obeys some distribution, which has three crucial features:

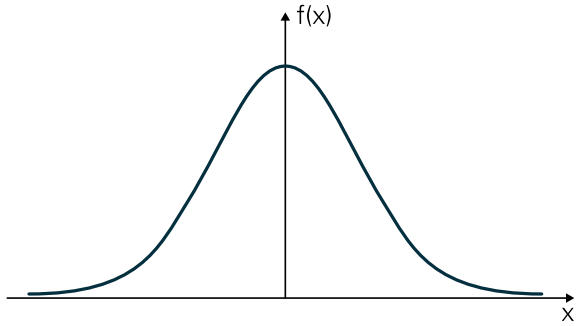
- shape,
- central location,
- variability.

Concerning the shape, the most common approach is to assume that the process parameter is normally distributed. Such a distribution is portrayed in Fig. 4.18. The Central location can be briefly perceived as an expected process parameter value whilst the variability signifies its spread around this value. Let us start with recalling the crucial measure of location, which is simply the mean or average value given by a well known formula:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad (4.4)$$

where x_i is the i -th process parameter observation whilst n stands for the number of observations and \bar{x} is the mean value. This measure can be easily calculated by forming the KPI incorporating the `Mean` command (cf. Appendix A). Another useful location measure is a median, which is the parameter value at which half of the observations fall above and half below. However, KIS.ME does not provide a direct command which can be used to calculate that. Fortunately, for that purpose, percentiles, implemented with the `Percentile` command, can be efficiently employed.

Fig. 4.18 Process parameter x versus its normal distribution function $f(x)$ (to do)



> **Percentile**

A percentile is simply defined as a statistical measure indicating the process parameter value below which a given percentage of observations in a group of observations fall. Thus, the 50th percentile is the value below and over which 50% of the process parameter observations can be found. As a result, the KPI calculating the median value can be simply implemented as $m = \text{Percentile}[x, 50]$, where m stands for the median whilst x is the set of observations being analysed. Note also that all observations x are at or below the 100th percentile, i.e., $m = \text{Percentile}[x, 100]$.

Let us proceed to variability measures. The most common ones are the standard deviation, the variance and the range. The first of those is defined as

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}, \tag{4.5}$$

where σ signifies the standard deviation, which can be calculated by defining the KPI involving $\text{sigma} = \text{Stdev}[x]$. The second variability measure is simply a square of the first one, i.e., σ^2 . Finally, the range is defined as the difference between the maximum and minimum values of $x \in \mathbb{X}$. Thus, the resulting KPI should be implemented with $\text{range} = \text{Max}[x, x] - \text{Min}[x, x]$. Having measures of location and variability, one can distinguished two possible SPC states of the process [1]:

Process in control: The process is in the state of control if it is subject to common cause variations only, which can be expected in any set of observations \mathbb{X} .

Process out of control The process is in the out-of-control state if it subject to both common and special cause variations. Thus, apart from the inevitable random and typical fluctuations, other unappealing factors affect process parameters.

A periodical evolution of a sample distribution of the process in control parameter is presented in Fig. 4.19. Even if a given process parameter is at an unacceptable level, e.g., FPY (4.3) is equal to 70%, one can be sure that such results can be expected for the subsequent time periods. Indeed, when one says that the process is in control,

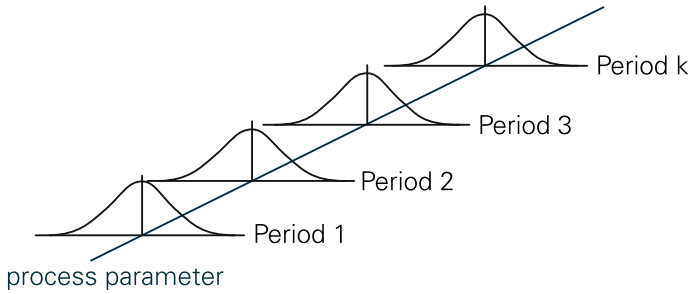


Fig. 4.19 Periodical evolution of the process-in-control parameter distribution

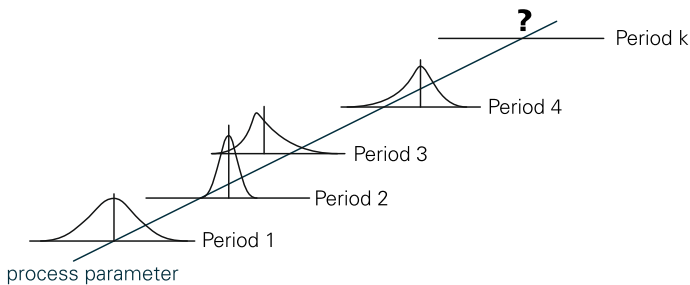


Fig. 4.20 Periodical evolution of the process-out-of-control parameter distribution

then this does not mean that it is working perfectly, but simply that it is predictable or periodically stable. Finally, improvements can be achieved by suitable rearrangement or optimization of the process. This is, however, beyond the scope of this section.

If the process parameter distribution changes with each periodically collected set of observations, then one has the process out of control. Such a situation is illustrated in Fig. 4.20. In practice, several processes obey such periodical behaviour. Thus, it is obvious that special cause variations should be identified and removed if possible. Similarly, the influence of common cause variations can be removed by a suitable analysis and improvement of the process.

Mean and standard deviation of assembly durations

Let us reconsider the example presented in Sect. 4.1.2 pertaining to monitoring task realization durations. The worker realizing such tasks is equipped with KIS.BOX Red indicating its current working mode, i.e., either assembly or idle periods. The objective is to implement KPIs calculating the mean and standard deviation of durations corresponding to the assembly mode. Moreover, the calculations should be realized within 15-min processing periods. As previously, the `button1ColorKpiDuration` Datapoint is used for calculation purposes whilst the results are visualized with the KPI Single Value widget. In particular, the widget should provide the maximum of the mean and standard deviation obtained during the last hour. The configuration of such a widget was described in Sect. 4.1.2,

and hence it is omitted. Before going to the KPI implementation, let us also recall that the KIS.BOX Red Button 1 operational LED is governed by two rules, which change its color between red (assembly mode) and green (idle mode) with the trigger associated with pressing the respective button. Thus, during the assembly process the `button1ColorKpiDuration` Datapoint returns 5 (red color) and 3 (green color). Let us proceed to the KPI implementation, which starts by defining x as an alias name of the `button1ColorKpiDuration` Datapoint. As a result, the KPIs calculating the mean and the standard deviation of the assembly durations are given by the following programs:

Mean of assembly durations

```
t=If[x==5,Duration[x],0];
y=Round[Mean[Filter[t>0]]/1000];
```

Standard deviation of assembly durations

```
t=If[x==5,Duration[x],0];
y=Round[Stdev[Filter[t>0]]/1000];
```

Finally, a sample of the performance of the above KPIs is presented in Fig. 4.21.

Taking into account the discussion presented in this section, the results in Fig. 4.21 clearly show that the assembly process is realized with greater duration variability (cf. Max standard deviation). This clearly means that such a process has to be suitably improved to get more predictable and less variable results. For that purpose, a set of widget control charts is to be introduced in the subsequent section.

4.3 Understanding process performance with widgets: A practical way to statistical control charts

The objective of this section is to provide a practical guidance on using four widget control charts:

- KPI Aggregated Chart;
- KPI Single Value Column;

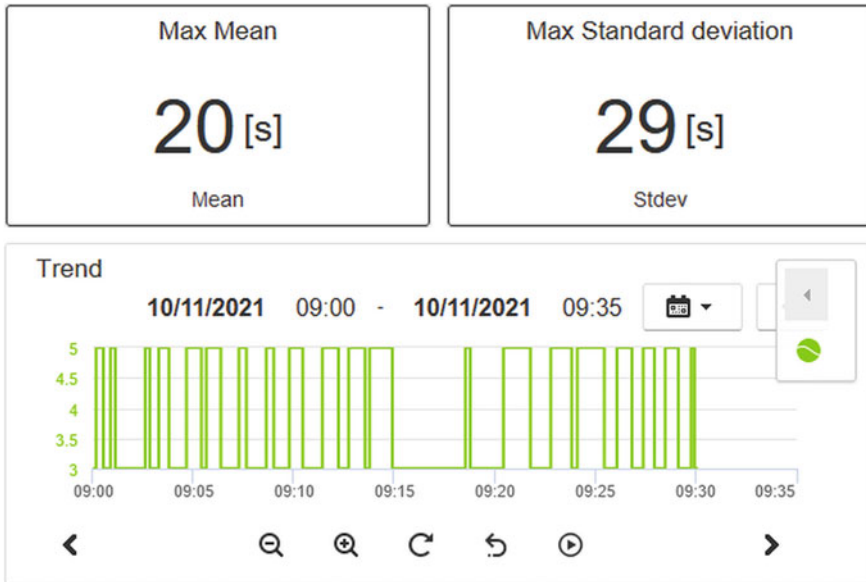


Fig. 4.21 Sample maximum mean and standard deviation of assembly durations

- KPI Pie Chart;
- KPI Single Period Chart.

They can be associated with any dashboard, which is located either in assets or asset groups (cf. Sect. 2.5). The first option can be accessed via Main menu → Assets → Edit dashboard → Add widget, which results in the view portrayed in Fig. 4.22. The second option is to follow Main menu → Asset groups and then select the desired workspace. Subsequently, Edit dashboard → Add widget have to be chosen, which yield the view presented in Fig. 4.23. Irrespective of the selected design approach, the quadruple of control chart widgets possesses the same functionality, which is to be discussed in the subsequent part of this section.

KIS.Device-based data generation

The control chart widgets being discussed in this section require data obtained with Datapoints of a given KIS.Device. Thus, to perform a unified presentation of all widgets, which can be easily reproduced, the following data generation approach is proposed:

Environment: It is defined with Workspace 1 and the dashboard of KIS.BOX Red.
Triggers: The triggers are based on the KIS.BOX Red Button 1 operational LED color, which can be either red or green. Additionally, each trigger is fired after one minute.

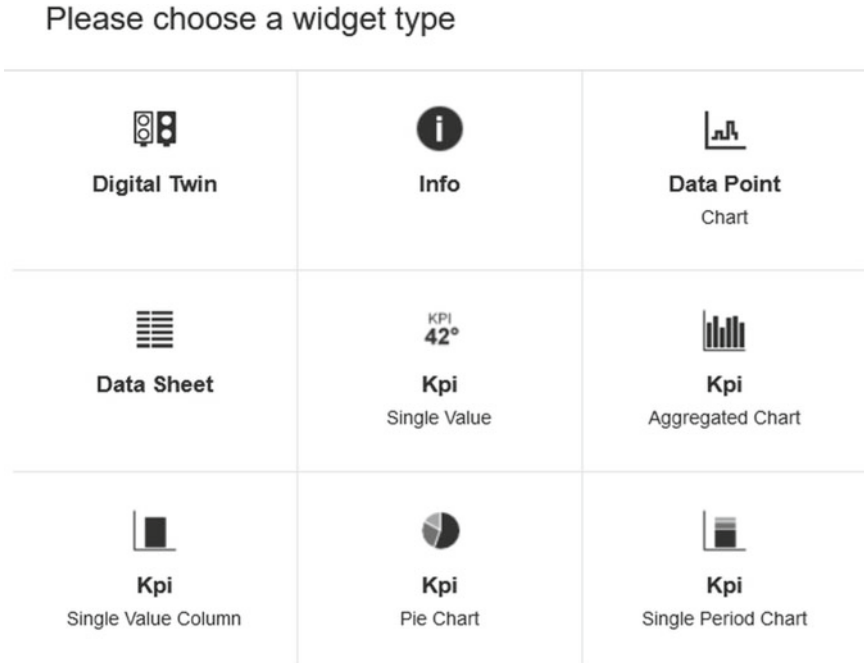


Fig. 4.22 KPI control charts within the asset dashboard

Actions: The associated actions are associated with changing the KIS.BOX Red Button 1 operational LED to an opposite value.

Initial condition: The KIS.BOX Red Button 1 operational LED is initialized with the associated digital twin (see Sect. 2.6), i.e., its color is set arbitrarily to either green or red.

The above functionality can be easily implemented using two rules, which are presented in Figs. 4.24 and 4.25. As a result of employing these, cyclically (every minute) changing values of the `button1ColorKpiDuration` Datapoint are generated, which are presented in Fig. 4.26. These data are to be used for statistical processing and visualization using the control chart widgets. Although the data presented in Fig. 4.26 looks like a uniform one with each cycle equal to one minute, there are some small discrepancies between cycle durations. This is related to the wireless transmission between the KIS.Device and KIS.MANAGER. Such behaviour is to be statistically analysed in the subsequent part of this section. This, however, requires defining suitable KPIs. In particular, it is proposed to employ two KPIs which will calculate mean values of the KIS.BOX Red Button 1 operational LED color durations over a 15-min processing period. Since there are only two colors, i.e., red and green, these KPIs are implemented as follows (with x being an alias of `button1ColorKpiDuration`):

Please choose a widget type











 <p>Floorplan A group floorplan</p>	 <p>Data Point Chart</p>	 <p>Data Sheet</p>
 <p>Aggregation</p>	 <p>KPI 42° Kpi Single Value</p>	 <p>Kpi Aggregated Chart</p>
 <p>Kpi Single Value Column</p>	 <p>Kpi Pie Chart</p>	 <p>Kpi Single Period Chart</p>

Fig. 4.23 KPI control charts within the asset group dashboard

Trigger 







Actions 






Fig. 4.24 First rule for KIS.BOX-based data generation

Mean of red color durations

```
t=If[x==5,Duration[x],0];
y=Round[Mean[Filter[t>0]]/1000];
```

Trigger i

2
1 KIS.BOX Red Button 1 Button 1 Color

After x minutes 1 i

Actions i

2
1 KIS.BOX Red Set LED Button 1 Color

Fig. 4.25 Second rule for KIS.BOX-based data generation

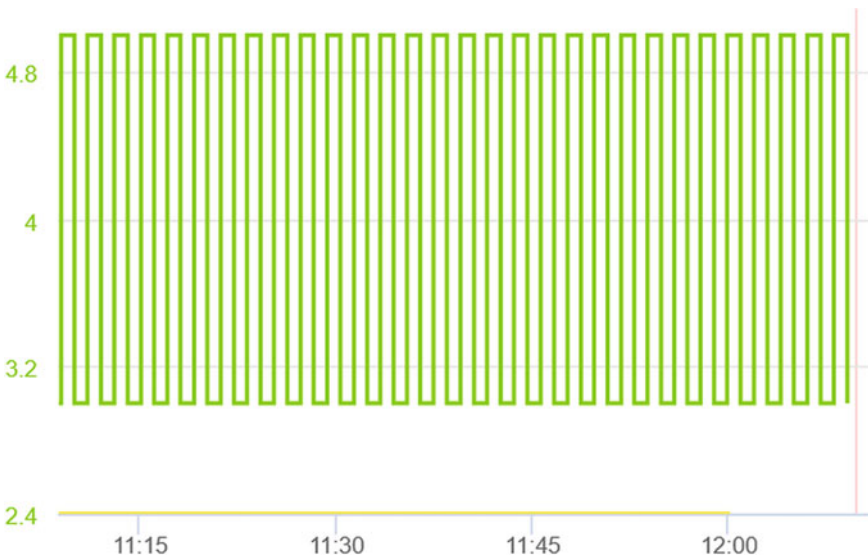


Fig. 4.26 KIS.BOX-based data sequence

Mean of green color durations

```
t=If[x==3,Duration[x],0];  
y=Round[Mean[Filter[t>0]]/1000];
```



4.3.1 Single Value Column Chart

The objective of this section is to provide practical guidelines for using the KPI Single Value Chart, which is simply a bar plot displaying an aggregated value of a selected KPI within a specified aggregation period (cf. Figure 4.2). This widget has the following properties:

Headline: the widget name, displayed on it;

Label: the name of the bar plot;

KPI name: the selected KPI name;

Unit: the KPI unit, e.g., seconds [s];

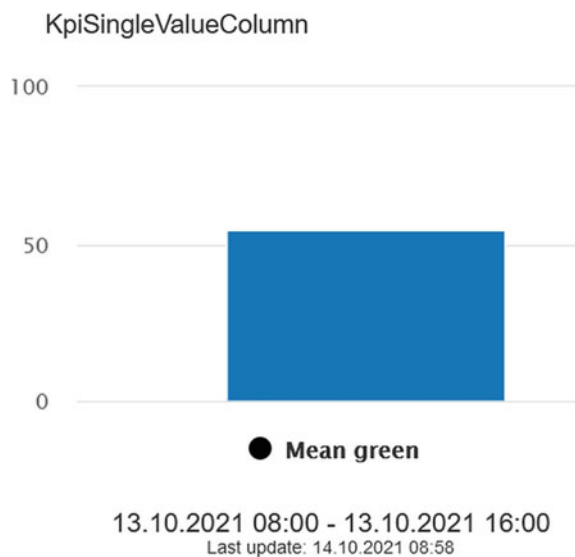
Date range: the selected, possibly historical, date and time range, e.g., a working shift;

Aggregation type: KPI values calculated within processing periods are aggregated within the date range using one of the following aggregations: sum, average, max, min (cf. Figure 4.2);

Zones: color zones associating the value range of the bar plot with the selected color.

Fig. 4.27 illustrates a sample bar plot. It displays the KPI values corresponding to the mean KIS.BOX Red Button 1 operational LED green color durations. They are aggregated using the average aggregation operator between 8:00–16:00 h on October 13th, 2021.

Fig. 4.27 Sample KPI single value column chart



4.3.2 Single Period and Pie Charts

The section aims at describing the way of using the KPI Single Period Chart, which is a multiple bar plot displaying an aggregated value of selected KPIs within a specified aggregation period (cf. Fig. 4.2). This widget has the following properties:

- Headline: the widget name, displayed on it;
- KPI name: the selected KPI names, whose values are used to form multiple bar plots;
- Label: the name of bar plot;
- Color: the color associated with the bar plot;
- Unit: the unit of the KPIs, e.g., seconds [s];
- Date range: The selected, possibly historical, date and time range, e.g., a working shift;
- Aggregation type: KPI values calculated within processing periods are aggregated within the date range using one of the following aggregations: sum, average, max, min (cf. Figure 4.2);
- Stack KPI: Defines the display form of bars (either stacked or separated).

Figure 4.28 presents a sample two bar plot. It displays two KPI values corresponding to the mean KIS.BOX Red Button 1 operational LED durations of either the green or red color states, respectively. They were aggregated using an average aggregation operator between 8:00–16:00 h on October 13th, 2021. Note that the same single period is used for all bar plots. Additionally, the units are defined, i.e., seconds [s] are used as units. Finally, the stacked version of the above plot is presented in

Fig. 4.28 Sample KPI single period chart

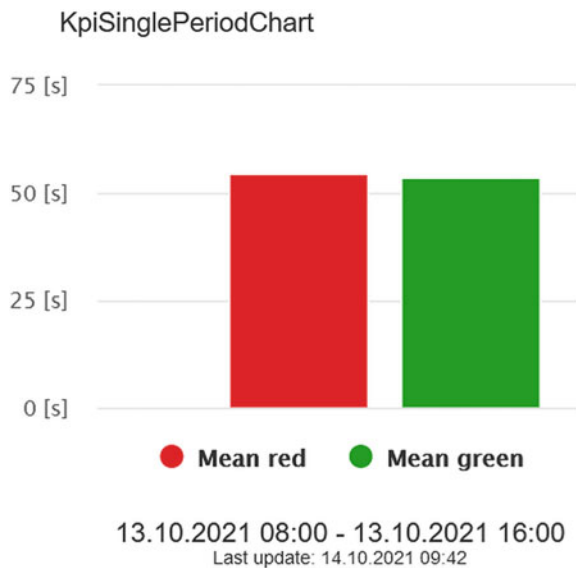


Fig. 4.29 Sample KPI single period chart with stacked bar plots

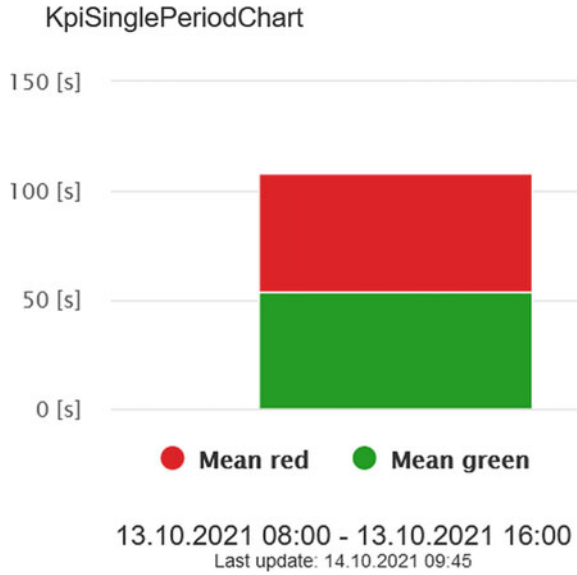


Fig. 4.30 Sample KPI pie chart

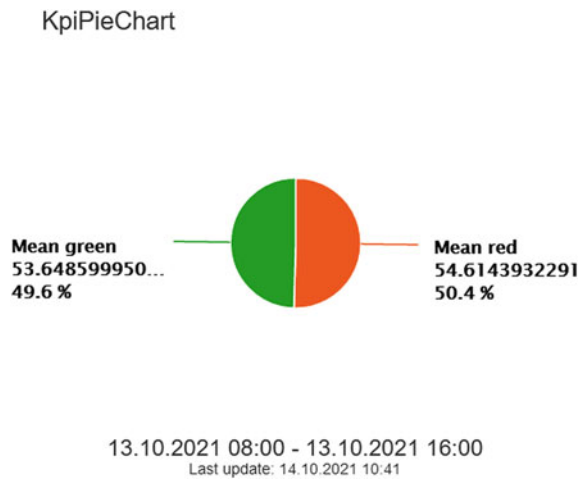


Fig. 4.29. Its appealing property is that it allows sum-based grouping of bar plots. Except for the bar plots, KIS.MANAGER provides a KPI pie chart. Apart from the Stack KPI, its properties are the same as those of the KPI Single Period Chart, but instead of using multiple bars a pie-like plot is employed. In particular, pieces of pies are proportional to the values of KPIs. As a result, the KPI pie chart counterpart of Fig. 4.29 is presented in Fig. 4.30. As can be observed, the percentage proportions between the pieces of the pie are displayed as well.

4.3.3 Aggregated Chart

The objective of this section is to show how to exploit the most advanced chart widget, i.e., the KPI Aggregated Chart. Contrarily to the charts presented in the preceding section, this one performs aggregations, which are grouped within a selected time period, e.g., every hour within a specified aggregation period. This widget has the following properties:

- Headline: the widget name, displayed on it;
- KPI name: the selected KPI names, whose values are used to form multiple plots;
- Label: the name of the plot;
- Color: the color associated with the plot;
- Unit: the KPI unit, e.g., seconds [s];
- Date range: the selected, possibly historical, date and time range, e.g., a working shift;
- Aggregation type: KPI values calculated within processing periods are aggregated within the date range using one of the following aggregations: sum, average, max, min (cf. Figure 4.2);
- Stack KPI: defines the display form of the plot either stacked or separated.
- Group by: the time spread between consecutive plots, e.g., one hour;
- Combined y-axis: determines if either a single or separated y-axis is used for all plots;
- Display as: separately determines the style of the plots, i.e., a bar, a line or filled triangles.

Figures 4.31 and 4.35 present the evolution of KPIs calculating the averaged mean durations of the KIS.BOX Red Button 1 operational LED, which can illuminate in

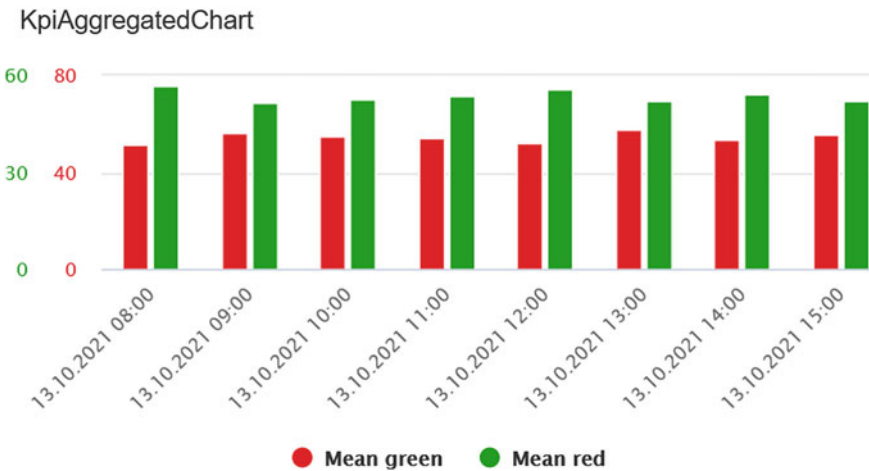


Fig. 4.31 Sample KPI aggregated chart

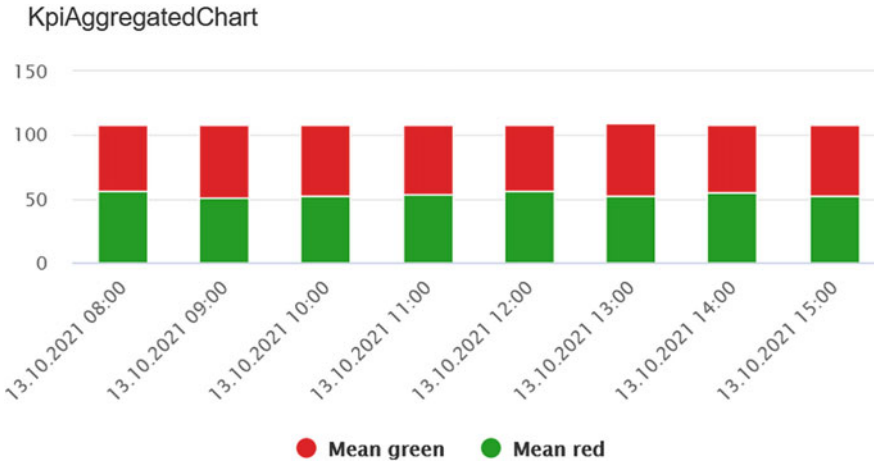


Fig. 4.32 Sample KPI aggregated chart with stacked bars

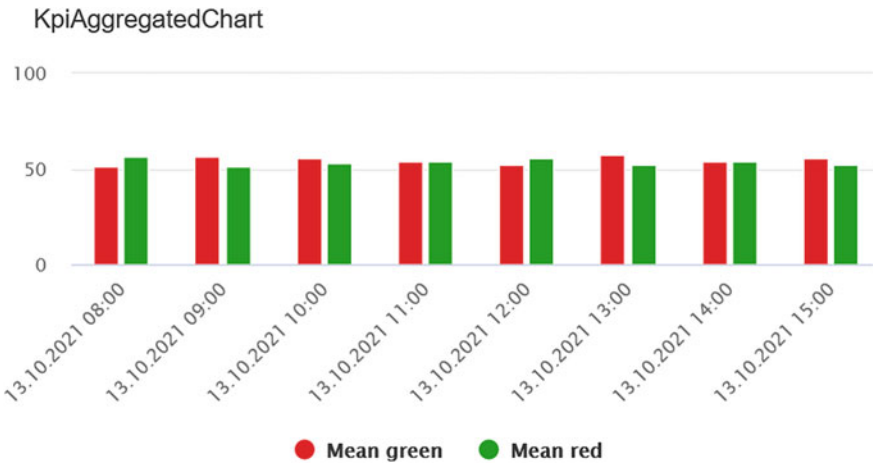


Fig. 4.33 Sample KPI aggregated chart with a single y-axis

either green or red, grouped by one hour. In particular, Fig. 4.31 presents a bar plot with individual y-axes. Figure 4.32 portrays the same data but with stacked bars. Subsequently, the data with a single y-axis is given in Fig. 4.33. Finally, Figs. 4.34 and 4.35 present the remaining plot options.

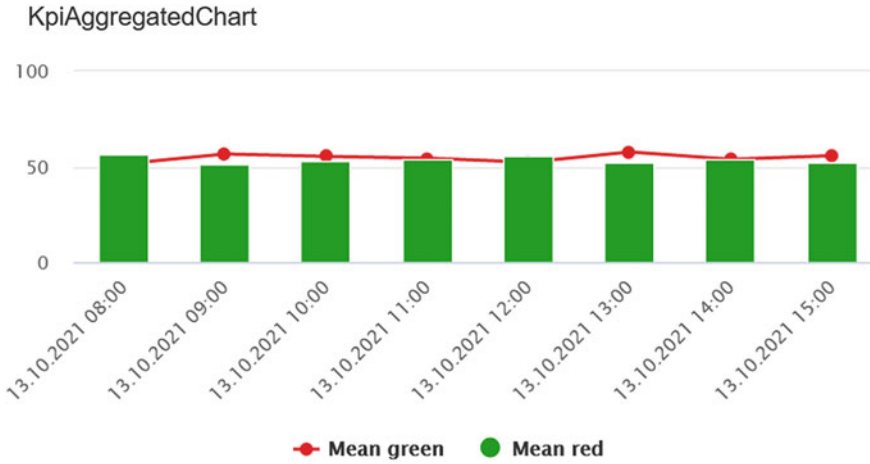


Fig. 4.34 Sample KPI aggregated chart with a line and bar plots

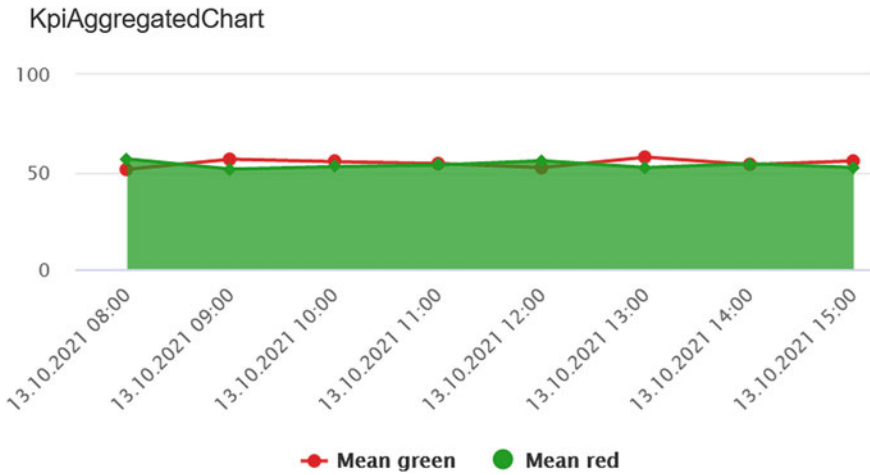


Fig. 4.35 Sample KPI aggregated chart with a line and filled triangle plots

4.4 Control Charts: Comparison and Analysis

As already mentioned in Sect. 4.2, SPC requires special measures to check if a given process is in the statistical control state. Thus, if a variable shaping process performance is described by the same distribution, then it is in the statistical control state.

The objective of this section is to introduce selected control charts, which are statistical tools that can be used for monitoring the process. As a result, they can provide suitable alerts pertaining to some disturbance acting on the process, leading to its out-of-control state. Finally, based on such results, one can find and minimize the disturbance cause.

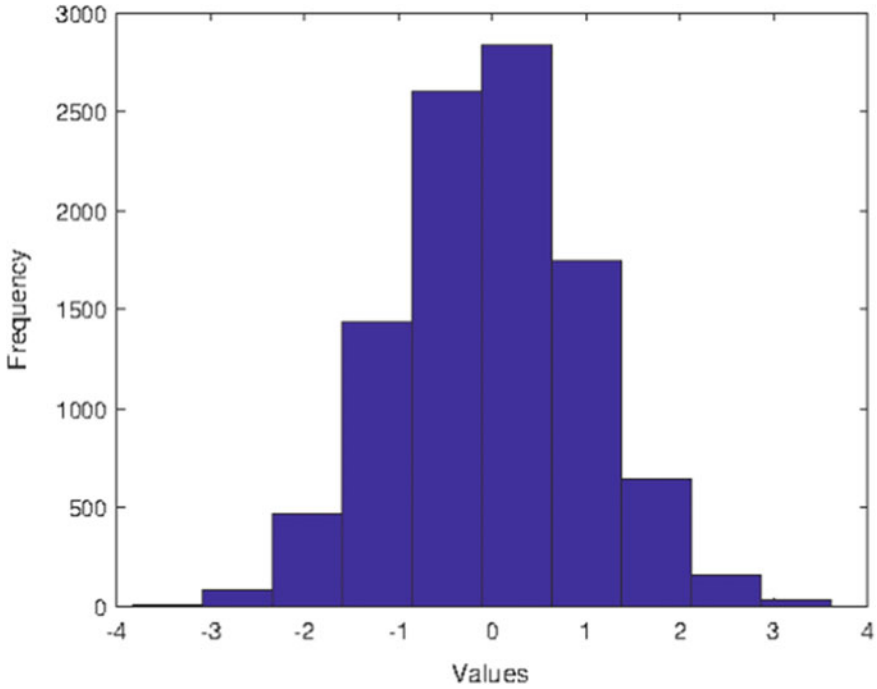


Fig. 4.36 Sample histogram

4.4.1 Histograms

A histogram can be perceived as a practical tool, which can be used for approximating the distribution of numerical data concerning a given variable (see Fig. 4.36). Its design boils down to dividing the range of values into equal intervals and then determining how many values are in each intervals. Such non-overlapping consecutive intervals of a variable are called bins. The most common approach is to assume that such bins (intervals) have equal widths. KIS.MANGER does not support histograms directly. Thus, the objective of this section is to provide a way of designing them. This can be realized in the following steps:

Step 1: Find the minimum and maximum values of a variable x , i.e., x_{\min} and x_{\max} .

Step 2: Select the number k and width w of histogram bins and then determine the bin ranges

$$x : x_{r,i+1} > x \geq x_{r,i}, \quad (4.6)$$

$$x_{r,i} = x_{\min} + (i - 1)w, \quad i = 1, \dots, k. \quad (4.7)$$

Step 3: For each bin, implement a KPI counting the values which fall inside it:

Histogram bin calculation

```
y=Count[Filter[x<xr1 && x>=xr]];
```

where $xr1$ stands for $x_{r,i+1}$ whilst xr signifies $x_{r,i}$.

Step 3: Visualize the KPIs using the KPI Single Period Chart and select summation as the aggregation method (see Sect. 4.3.2).

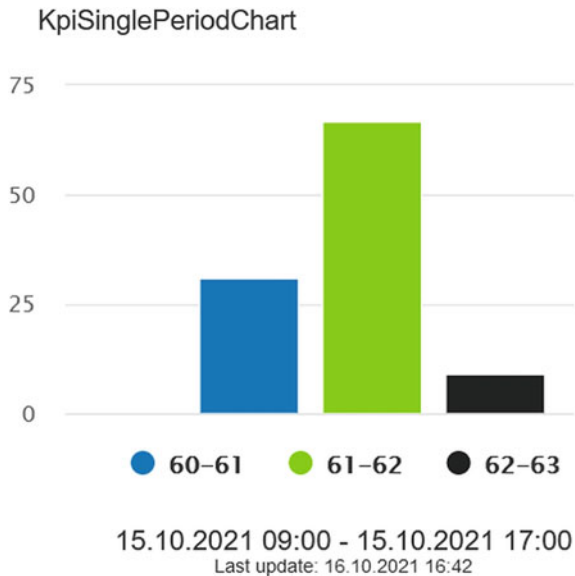
> **Grouping histograms**

The above approach allows preparing histograms displaying data distribution inside a selected aggregation. However, by replacing the KPI Single Period Chart with the KPI Aggregated Chart (see Sect. 4.3.3), one can observe an evolution of the data distribution grouped by a specified time period, i.e., an hour or a day.

Figure 4.37 presents the histogram concerning red color durations of the KIS.BOX Red operational LED. A sample bin of this histogram is implemented as follows:

```
t=If[x==3,Duration[x]/1000,0];
y=Count[Filter[t>=61 && t<62]];
```

Fig. 4.37 KPI single period chart-based histogram



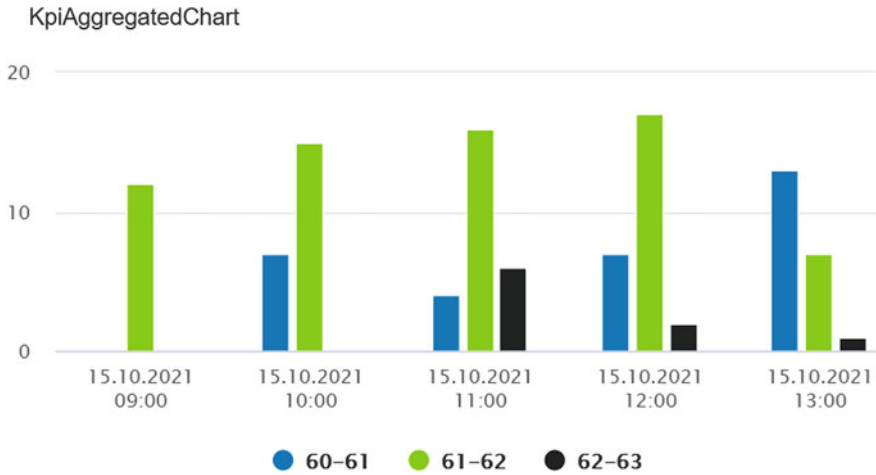


Fig. 4.38 KPI aggregated chart-based evolving histogram

For the sake of presentational simplicity, the histogram has three bins only. An evolving histogram, which is grouped by hours, is presented in Fig. 4.38.

➤ Duration versus the processing period

Each KPI is periodically calculated within a given processing period, which can be set at 15, 30 or 60 min. Moreover, any KPI calculation is started at an full hour, e.g., 9.00 h. Let us reconsider the process of changing the KIS.BOX Red operational LED color every minute, i.e., after one minute the color is changed. Figure 4.30 clearly shows that the mean values of the durations corresponding to either the red or green color is not so close to 60 s. The reason behind such a situation is presented in Fig. 4.39. In this sample case, it can be observed that the KPI calculation ends at 10.00 h, and hence the last red color state duration (red corresponds to the numerical 5) is lower than 60 s. Such cases, as well as similar ones, which may happen at the beginning of the KPI calculation, have considerable influence on both the mean and the standard deviation. One of possible remedies is to eliminate improbable values using the `Filter[]` command and then calculate the mean and the standard deviation. One can also use different measures, e.g., the proportion between the sum of durations and the processing period time, which can be obtained with

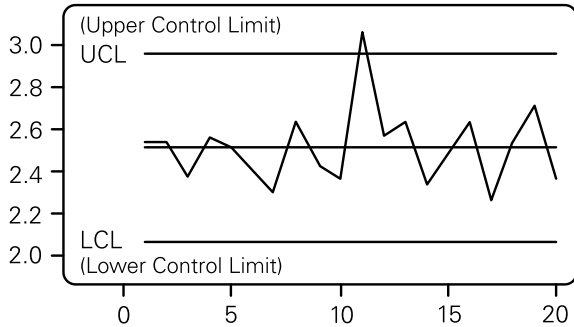
```
s=Sum[If[x==5,Duration[x],0]]/Interval[];
```

Nevertheless, appropriate selection of the processing period is crucial for the correctness of the achieved results.



Fig. 4.39 Duration versus processing period

Fig. 4.40 Sample control chart



4.4.2 Control Charts with Limits

The objective of this section is to introduce crucial control charts:

- \bar{x} : the mean-based chart,
- R : the range-based chart,
- p : the proportion-based chart.

There are, of course, plenty of different control charts. However, their presentation is beyond the scope of this book, and hence the reader is referred to [1] for a comprehensive explanation. Irrespective of the control chart type, they are used to visualize the evolution of some statistical measure against the so-called lower and upper control limits (LCLs and UCLs). Such a process is portrayed in Fig. 4.40. Apart from the above limit one can observe the center line, which is plotted at the level of the expected value of a given statistic. The only way to plot control charts in KIS.ME is to use the KPI Aggregated Chart, which can group the displayed data within a specified period, i.e., an hour or a day (cf. Sect. 4.3.3). Apart from the grouping

period, it is suggested to select an appropriate KPI processing period, which should be lower than the grouping one. Thus, by selecting the average aggregation method in the KPI Aggregated Chart, the mean of a given statistics can be calculated. However, before proceeding to the practical implementation, let us provide calculation rules for all above-listed charts. Let us start with the \bar{x} chart, for which the central line $\bar{\bar{x}}$ can be estimated using the data from previous processing period or can be assumed to be known. Subsequently, the control limits can be implemented with the well known three standard deviations rule. For that purpose, it should be recalled that the standard deviation of the mean is related to the standard deviation of the process variable σ in the following way: $\sigma_{\bar{x}} = \sigma/\sqrt{n}$, where n is the sample size within the processing period. Note that in usual conditions the $\sigma_{\bar{x}}$ has to be estimated, which can be realized using, e.g., one of the following formulas:

$$\hat{\sigma}_{\bar{x}} = \frac{1}{d_1} \frac{\bar{s}}{\sqrt{n}}, \quad (4.8)$$

$$\hat{\sigma}_{\bar{x}} = \frac{1}{d_2} \frac{\bar{r}}{\sqrt{n}}, \quad (4.9)$$

where

- \bar{s} is the mean standard deviation calculated over m processing periods;
- \bar{r} is the mean range calculated over m processing periods.

Moreover, d_1 and d_2 are known constants, which are given in Table 4.4. This can, of course, be realized under the assumption that the process is in the control state for a set of m processing periods being considered. Thus, the \bar{x} chart can be summarized as follows:

$$\begin{aligned} \text{Center line: } & \bar{\bar{x}}, \\ \text{UCL: } & \bar{\bar{x}} + 3\hat{\sigma}_{\bar{x}}, \\ \text{LCL: } & \bar{\bar{x}} - 3\hat{\sigma}_{\bar{x}}. \end{aligned}$$

Using a similar line of reasoning, the \bar{R} chart can be described:

$$\begin{aligned} \text{Center line: } & \bar{\bar{r}}, \\ \text{UCL: } & D_4\bar{\bar{r}}, \\ \text{LCL: } & D_3\bar{\bar{r}}, \end{aligned}$$

where m_r stands for the mean of the range of n measurements of the process variable while D_3 and D_4 are known constants, which are given in Table 4.4. Note that the constants for larger n are available, e.g., in [1]. Let us proceed to the control charts for proportions, which are widely known as p charts. One of the popular examples of using the p chart is to visualize the ratio between the number of defective items and their total (cf. 4.3). Similarly as in the previous cases, a natural candidate for the central line is the mean \bar{p} proportion calculated over m processing periods. Let us also recall that the process used for the calculation of \bar{p} has to be in the statistical control state. Finally, the p chart can be summarized as follows:

Table 4.4 Constants for control chart design

n	d_1	d_2	D_3	D_4
2	1.7725	1.1284	0.0000	3.2665
3	1.3820	1.6926	0.0000	2.5746
4	1.2533	2.0588	0.0000	2.2820
5	1.1894	2.3259	0.0000	2.1145
6	1.1512	2.5344	0.0000	2.0038
7	1.1259	2.7044	0.0757	1.9243
8	1.1078	2.8472	0.1362	1.8638
9	1.0942	2.9700	0.1840	1.8160

Center line: \bar{p} ,

$$UCL: \bar{p} + 3\sigma_p = \bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}},$$

$$LCL: \bar{p} - 3\sigma_p = \bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}.$$

Having all necessary ingredients, let us proceed to the implementation of the above control charts. We start with the \bar{x} chart. The variable being controlled is called x . Thus, the x chart can be implemented using the following steps:

Step 1: Determine \bar{x} (used as $xbar$), set up a new KPI and give it the name CL $xbar$, whilst its implementation is realized as follows:

$$CL=Max[xbar, xbar];$$

Step 2: Using either (4.8) or (4.9), determine $\hat{\sigma}_x$ (used as $sigmax$) and then set up two KPIs (UCL $xbar$ and LCL $xbar$). Implement the LCL and UCL KPIs with the following programs:

$$UCL=Max[xbar+3*sigmax, xbar+3*sigmax];$$

$$LCL=Max[xbar-3*sigmax, xbar-3*sigmax];$$

Step 3: Set up a new KPI and give it the name m_x , along with the implementation shaped by

$$meanx=Mean[x];$$

Step 4: Visualize CL $xbar$, LCL $xbar$, UCL $xbar$ and m_x with the KPI Aggregated Chart.

The implementation of the R chart can be realized in a similar way:

Step 1: Determine \bar{r} (used as br), and then set up a new KPI and give it the name CLR, whilst its implementation is realized as follows:

$$rbar=Max[br, br];$$

Step 2: Set up two KPIs (UCLR and LCLR). Implement the LCL and UCL KPIs with the following programs:

```
UCL=D4*Max[br,br];
```

```
LCL=D3*Max[br,br];
```

Step 3: Set up the new KPI and give it the name mr, along with the implementation shaped by

```
meanr=Max[x,x]-Min[x,x];
```

Step 4: Visualize CLR, LCLR, UCLR and mr with the KPI Aggregated Chart.

Assuming that the variable p can be calculated, e.g., in a similar way as (4.3), the implementation of the p chart boils down to the following:

Step 1: Determine \bar{p} (used as bp), set up a new KPI and give it the name CL_p, whilst its implementation is realized as follows:

```
c1p=Max[bp,bp];
```

Step 2: Determine $\sigma_p = \sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$ (used as sigmap) and set up two KPIs (UCL_p and LCL_p). Implement the LCL and UCL KPIs with the following programs:

```
UCL=Max[bp+sigmap,bp+sigmap];
```

```
LCL=Max[bp-sigmap,bp-sigmap,0];
```

Step 3: Set up a new KPI and give it the name mp, along with the implementation shaped by

```
meanp=p;
```

Step 4: Visualize CL_p, LCL_p, UCL_p and mp with the KPI Aggregated Chart.

The objective of this section was to provide recipes for calculating essential control charts with KIS.ME. Note that it is mandatory to use aggregation functions inside a KPI implementation. Thus, the `Max[variable, variable]` command is employed, which in the case of a scalar variable returns simply its value.

4.5 Practical Example Revisited

Let us reconsider the data generation example introduced in Sect. 4.3. It pertains to alternately changing the KIS.BOX Red operational LED color from green to red. Each change is realized after one minute. The objective of this section is to provide dedicated implementations which can be directly used for this particular case.

Let us start with the implementation of the \bar{x} chart. Using the historical data from a set of processing periods, the mean $\bar{\bar{x}} = 61[s]$ and the mean range $\bar{r} = 1.5[s]$

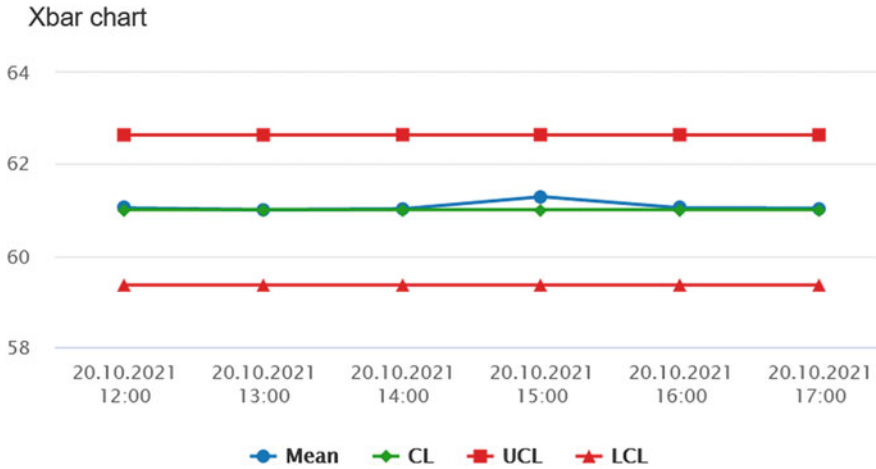


Fig. 4.41 \bar{x} chart

were obtained. Thus, applying these values to (4.9) with $n = 7$ yields $\sigma_{\bar{x}} = 0.547[s]$. Having these values, four KIPs were implemented:

CLxbar:

```
xbar=61;
CL=Max[xbar, xbar];
```

UCLxbar:

```
xbar=61; sigmax=0.547;
UCL=Max[xbar+3*sigmax, xbar+3*sigmax];
```

LCLxbar:

```
xbar=61; sigmax=0.547;
LCL=Max[xbar-3*sigmax, xbar-3*sigmax];
```

mx:

```
d=If[x==5, Duration[x], 0];
y=Mean[Filter[d>60000]]/1000;
```

The values of these KPIs are calculated with a 15-min processing period, whilst the KPI Aggregated Chart is grouping and averaging their values every hour. The resulting \bar{x} chart is presented in Fig. 4.41. As can be observed, the mean values (blue line) are inside the control limits (red lines).

Let us proceed to the second chart, i.e., R . Using Table 4.4 for $n = 7$ gives $D_3 = 0.0757$ and $D_4 = 1.9249$. Having these values, four KIPs were implemented:

CLR:

```
rbar=1.5;
CL=Max[rbar, rbar];
```

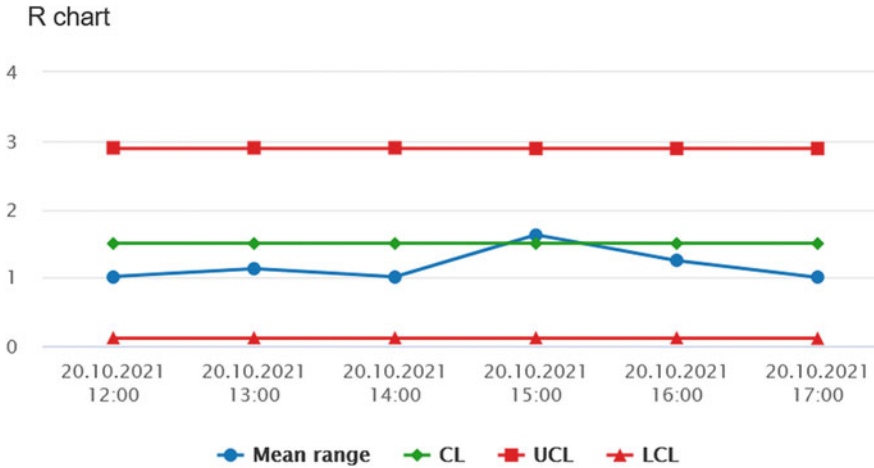


Fig. 4.42 R chart

UCLR:

```
rbar=1.5;
CL=1.9249*Max[rbar,rbar];
```

LCLR:

```
rbar=1.5;
CL=0.0757*Max[rbar,rbar];
```

mr:

```
d=If[x==5,Duration[x],0];
b=Filter[d>60000]/1000;
y=Max[b,b]-Min[b,b];
```

The values of these KPIs are calculated with a 15-min processing period, whilst the KPI Aggregated Chart is grouping and averaging their values every hour. The resulting R chart is presented in Fig. 4.42. As can be observed, the mean range fluctuates in a more intense way than the mean value. However, it is close to the mean range (green line) and within the control limits (red lines).

Let us process to the last chart, i.e., p . The proportion being considered is defined as the ratio between the sum of red color durations and that of both red and green color durations. In this case, with 15-min processing period, one should expect (on average) $n = 15$, which corresponds to changing the color every minute. As a result, $\sigma_p = 0.129$ while $\bar{p} = 0.5$. Having these parameters, the implementation of the p chart boils down to programming the following KPIs:

CLp:

```
pbar=0.5;
CL=Max[pbar,pbar];
```

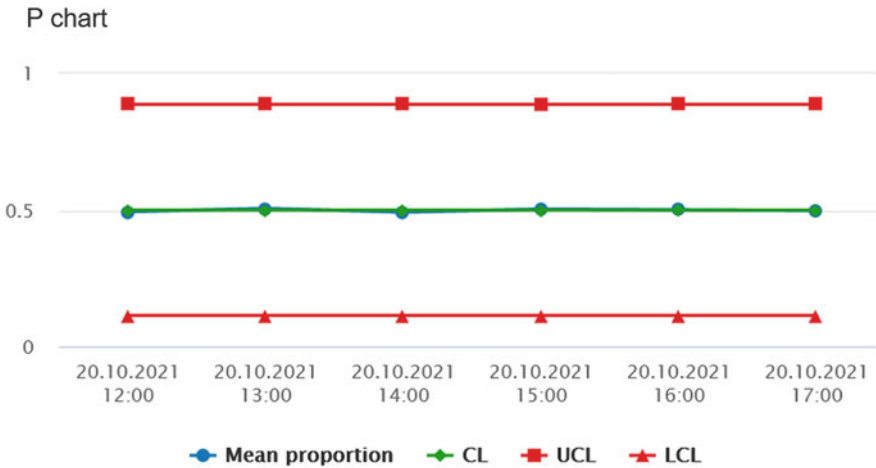


Fig. 4.43 *p* chart

UCL_p:

```
pbar=0.5; sigmap=0.129;
UCL=Max[pbar+3*sigmap,pbar+3*sigmap];
```

LCL_p:

```
pbar=0.5; sigmap=0.129;
LCL=Max[pbar-3*sigmap,pbar-3*sigmap];
```

mp:

```
dred=If[x==5,Duration[x],0];
yred=Sum[Filter[dred>60000]];
dgreen=If[x==3,Duration[x],0];
ygreen=Sum[Filter[dgreen>60000]];
y=yred/(ygreen+yred);
```

The values of these KPIs are calculated with a 15-min processing period, whilst the KPI Aggregated Chart is grouping and averaging those every hour. The resulting *p* chart is presented in Fig. 4.43.

As can be observed, the mean values (blue line) are inside the control limits (red lines).

4.6 Training Exercises

4.1 KIS.Device data generation revisited

Exercise requirements: The exercise requires access to one KIS.LIGHT.

1. Similarly as in Sect. 4.3, implement a set of rules changing the KIS.LIGHT operational LED color from red to green. Both periods should be equal to one minute.

2. Use the Datapoint chart to visualize KIS.LIGHT behaviour.
3. Initialize the KIS.LIGHT color using its digital twin (cf. Sect. 2.6).

4.2 Your own way of calculating the mean and the standard deviation

Exercise requirements: The exercise requires access to one KIS.LIGHT and completion of Exc. 4.1.

1. Without using `Mean[]` and `StdDev[]`, according to (4.4) and (4.5), implement your own KPIs calculating the mean and standard deviations of KIS.LIGHT operational LED red color durations. Ensure also that all durations shorter than 60s are appropriately filtered and that the KPI processing period is equal to 15 min.
2. Use the KPI Single Period Chart to visualize the maximum values of the implemented KPIs within the selected aggregation period.
3. use the KPI Aggregated Chart to visualize the maximum values of the implemented KPIs, which are grouped by the hour within the aggregation period.

4.3 Median assembly time

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Let us consider a single container that is fed to the assembly system. In particular, the container includes the following parts (digits):

$$C = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. \quad (4.10)$$

2. The assembly task boils down to writing down the subsequent digits from the container. Thus, if all letters are used, then the container is empty and another one can be processed assuming that it is available.
3. Implement a set of rules ensuring that the KIS.BOX Button 1 operational LED is red during the assembly process and green otherwise. Associate triggers of these rules with pressing KIS.BOX Button 1.
4. Implement a KPI providing information about median assembly time.
5. Use a set of selected widgets to visualize the obtained results.

4.4 Histograms

Exercise requirements: The exercise requires access to one KIS.LIGHT and completion of Exc. 4.1.

1. According to the approach presented in Sect. 4.4.1, implement KPIs calculating 10 equally sized histogram bins spread over the interval of [60, 65] seconds (use a 15-min processing period).
2. Use the KPI Single Period Chart to visualize the histogram.
3. Use the KIS.LIGHT digital twin to interrupt the automatic color changing of the KIS.LIGHT operational LED, i.e., hold its red color for one to three seconds from time to time.
4. Analyze and explain the obtained results.

4.5 Control charts

Exercise requirements: The exercise requires access to one KIS.LIGHT and completion of Exc. 4.4.

1. According to the approach presented in the Sect. 4.4.2, implement the \bar{x} and R charts;
2. Use the KIS.LIGHT digital twin to interrupt the automatic color changing of the KIS.LIGHT operational LED, i.e., hold its red color for one to three seconds from time to time.
3. Analyze and explain the obtained results.

4.6 Sharing KPIs

Exercise requirements: The exercise requires access to two KIS.LIGHTs (assigned to the same workspace) and completion of Exc. 4.5.

1. Share all KPIs associated with the KIS.LIGHT used in Exc. 4.5 with the second KIS.LIGHT.
2. Implement the \bar{x} and R charts for the first and second KIS.LIGHT within their common workspace.
3. Compare and analyze control charts of both KIS.LIGHTs.

4.7 CDP calculation: Idle working time

Exercise requirements: The exercise requires access to two KIS.BOXes (assigned to the same workspace).

1. Let us reconsider two workers performing identical tasks at a single assembly station. Both of them use KIS.BOX 1 to indicate two states:

Assembly in progress: Exemplified by the red color of operational LEDs,
 Idle: Exemplified by the green color of operational LEDs.

This means that Worker 1 uses KIS.BOX 1 Button 1 while Worker 2 utilizes KIS.BOX 1 Button 2.

2. Implement the switching rules between assembly and idle states (cf. Figs. 4.3 and 4.4).
3. Repeat the above implementation for KIS.BOX 2, which is associated with a second pair of workers.
4. Using the approach presented in Fig. 4.9, implement CDPs indicating the idle state in each group.
5. For each worker group, i.e., each KIS.BOX, implement KPIs calculating the mean and the median idle time along with its standard deviation over 15-min processing periods.
6. Use a set of selected widgets to visualize and analyze the obtained results.

4.7 Concluding Remarks

The preliminary objective of this chapter was to introduce the concepts of calculating Datapoints and key performance indicators. Both of them can be used for processing data, but in a completely different way. Indeed, CDPs do so in a static way without taking data history into account. Contrarily, KPIs operate within specific processing periods. This means that the entire processing period data set is used for calculating the KPI value. For that purpose, aggregation functions can be used. The periodical values of KPIs can be further aggregated within an arbitrary period. This can be achieved with a set of various KPI chart widgets, which were carefully described. As a result, essential statistical analysis can be realized with the well-known measures of location and variability. Apart from these numerical values, it was also shown how to implement histograms. The final part of the chapter discussed the concept of control charts, which can be efficiently used to keep the process under statistical control. The chapter was concluded with a set of training exercises, which validate the knowledge gathered within it.

References

1. T. Stapenhurst, *Mastering Statistical Process Control* (Elsevier, Amsterdam, 2013)
2. D.C. Montgomery, *Introduction to Statistical Quality Control* (John Wiley & Sons, London, 2020)
3. J.S. Oakland, *Statistical Process Control* (Routledge, London, 2007)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 5

Mastering System Monitoring and Control



5.1 Defining the Performance Cost Function and Its Control

The objective of this section is to define the performance cost function for the multiple point–single transporter system described in Sect. 3.3. The process starts with providing the availability of the transportation system. Subsequently, the performance cost function is defined over a set of routs. Thus, the entire performance is related with the sum of all cost functions divided by the total run time.

Let us start defining the work environment for the transporter operator:

Routes: The transporter operates within Workspace 1 according to the routs presented in Fig. 5.3.

KIS.Device: The transporter operator uses KIS.BOX Transporter to signify the start and finish of a given transportation task.

Route assignment: The route assignment is realized by the KIS.MANAGER operator, which is setting KIS.Box Transporter Button 1 operational LED color to the one corresponding to the desired route.

> System operators

It is important to stress the fact that there are two operators within the system:

KIS.MANAGER operator: A human being assigning a transport task within a selected route and monitoring its realization.

Transporter operator: A human being physically realizing transportation tasks.

Table 5.1 Set of possible routes

Route	Route color	Points
1	Blue	S2→A3→S1→A2→A1→S1→S2
2	Turquoise	S1→A1→S1
3	Green	S2→S1→A2→S1→S2
4	Magenta	S2→A3→S2
5	Red	S2→A3→S1→A2→S1→S2
6	Black	Idle state

S: supermarket point, A: assembly point

Let us start with implementing a single route system, i.e., a blue one, and calculating transporter availability, which is defined as

$$\text{Availability} = \frac{\text{Run time}}{\text{Planned transportation time}}, \quad (5.1)$$

where the run time is the amount of time spent on the transportation whilst the planned transportation time is obtained as follows:

$$\text{Planned transportation time} = \text{Shift length} - \text{breaks} \quad (5.2)$$

and signifies the time span in which the transporter can be operational. As an example, let us consider an eight-hour shift with a one-hour break, which gives

$$\text{Planned transportation time} = 480 - 60 = 420(\text{min}). \quad (5.3)$$

As can be observed in Table 5.1, the idle state corresponds to the time in which the transporter is not realizing any transportation. However, before proceeding to the implementation, let us define the transportation acknowledgement and realization procedure:

1. The KIS.MANAGER operator sets the color of the KIS.BOX Transporter Button 1 operational LED to the one corresponding to a desired route.
2. The transporter operator acknowledges and starts the transportation action by pressing KIS.BOX Transporter Button 2.
3. The KIS.BOX Transporter Button 2 operational LED changes its color to the same as the one of the KIS.BOX Transporter Button 1 operational LED.
4. The KIS.BOX Transporter Button 1 operational LED color is changed to black.
5. The transporter operator accomplishes the transportation and then presses KIS.BOX Transporter Button 2.
6. The KIS.BOX Transporter Button 2 operational LED color is changed to black.

Trigger **i**


KIS.BOX Transporter
Button 2
Pressed

Conditions **i**


KIS.BOX Transporter
Button 1
Button 1 Color
EQUAL


Actions **i**


KIS.BOX Transporter
Set LED
Button 1 Color

 Flashing


KIS.BOX Transporter
Set LED
Button 2 Color

 Flashing

Fig. 5.1 Sample rule for acknowledging Route 1

Note that, after Step 4 of the above procedure, the KIS.MANAGER operator can arrange a consecutive route by setting the KIS.BOX Transporter Button 1 operational LED color. The implementation of Steps 1–4 can be realized with five rules like the sample one for the blue route, which is given in Fig. 5.1. The rules for different routes can be implemented in a similar way, and hence they are omitted. Finally, the rule implementing Steps 6–7 is given in Fig. 5.2. It is also assumed that each route has an associated ideal route time. These times should be perceived as optimal performance goals on a given route (see Table 5.2). Having these data, one can define a performance cost function over the processing period:

$$J = \frac{1}{t_r} \sum_{i=1}^{n_r} n_i t_i, \tag{5.4}$$

where t_r is the run time within the processing period, $n_r = 5$ is the number of routes (cf. Table 5.2), t_i is the ideal transportation time of the i -th route while n_i is the number of cycles on the i -th route. Thus, in the ideal case, this the above function should be equal to 1, which signifies perfect performance of the transporter.

Let us start with the KPI which can be used for the calculation of (5.1) with KIS.BOX Transporter. For that purpose, it is assumed that the planned transportation time is equal to 420 min (cf. 5.3):

Trigger Conditions Actions 

Fig. 5.2 Rule for accomplishing the transportation action

Table 5.2 Routes and ideal route times

Route	Route color	Ideal route time (min)
1	Blue	11
2	Turquoise	4
3	Green	8
4	Magenta	3
5	Red	9
6	Black	–

$$d = \text{If}[\text{Not}[x == 2], \text{Duration}[x], 0];$$

$$y = \text{Sum}[d] / 60000 / 420;$$

where x is an alias name of `button2ColorKpiDuration`. Note that the KPI starts with determining all durations for a non-idle state, which is equivalent to the black color of the KIS.BOX Transporter Button 2 operational LED. To get a fair assessment of (5.1), the results provided by the above KPI should be aggregated within an eight-hour shift using the sum aggregation mechanism. This can be achieved with the KPI Aggregated Chart, which can group the results in a selected aggregation period, i.e., an hour or a day. An alternative approach is to calculate the availability within a given processing period, which can be realized as follows:

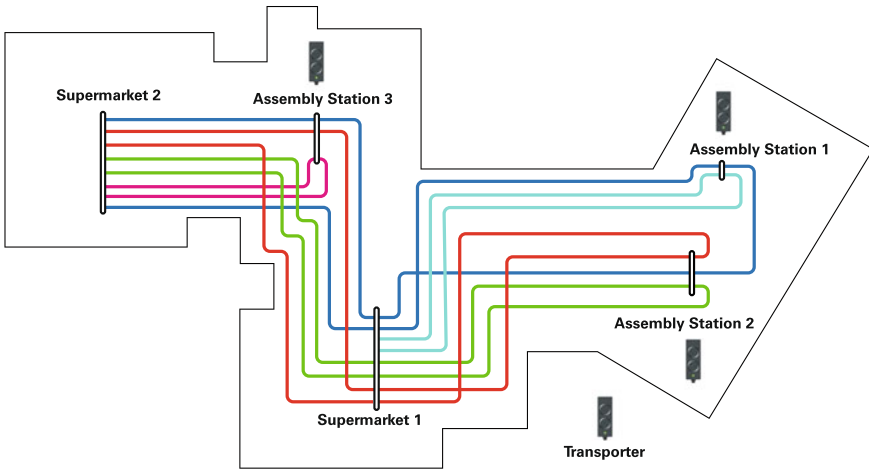


Fig. 5.3 Floorplan with a set of routes

```
d=If [Not [x==2] , Duration [x] , 0] ;
y=Sum [d] / Interval [ ] ;
```

Finally, an average availability can be calculated and visualized using KPI charts. Let us proceed to the implementation of (5.4), which should be started with introducing $n_r = 5$ ideal transportation times (cf. Table 5.2):

```
t1=11;
t2=4;
t3=8;
t4=3;
t5=9;
```

The next step boils down to calculating the number of cycles on a given route $n_i, i = 1, \dots, n_r$. For that purpose let us recall the numerical counterparts of colors, which are given in Table 2.2. The implementation consists in calculating the number of KIS.BOX Transporter Button 2 operational LED color changes. This can be realized using `led2ColorKpi` with an associated alias variable y . As a result, the KPI code can be extended as follows:

```
t1=11;
t2=4;
t3=8;
t4=3;
t5=9;
```

```

n1=Sum [ If [ y==0 , 1 , 0 ] ] ;
n2=Sum [ If [ y==1 , 1 , 0 ] ] ;
n3=Sum [ If [ y==3 , 1 , 0 ] ] ;
n4=Sum [ If [ y==4 , 1 , 0 ] ] ;
n5=Sum [ If [ y==5 , 1 , 0 ] ] ;

```

The last stage of the implementation is to calculate the run time t_r and the current value of the performance function (5.4):

```

t1=11 ;
t2=4 ;
t3=8 ;
t4=3 ;
t5=9 ;
n1=Sum [ If [ y==0 , 1 , 0 ] ] ;
n2=Sum [ If [ y==1 , 1 , 0 ] ] ;
n3=Sum [ If [ y==3 , 1 , 0 ] ] ;
n4=Sum [ If [ y==4 , 1 , 0 ] ] ;
n5=Sum [ If [ y==5 , 1 , 0 ] ] ;
d=If [ Not [ x==2 ] , Duration [ x ] , 0 ] ;
tr=Sum [ d ] / 60000 ;
J=1 / tr * ( t1 * n1 + t2 * n2 + t3 * n3 + t4 * n4 + t5 * n5 ) ;

```

Let us consider a sample transportation request sequence, which is presented in Fig. 5.4. In particular, there are 15 transportation requests, which are ordered by the KIS.MANAGER operator using the KIS.BOX Transporter Button 1 operational LED color (cf. Sect. 2.6). Association of a given route with the assembly stations (A1–A3) is depicted as well. Using Table 5.2 and Fig. 5.4, one easily see that there are four cycles for route 1, three for route 2, three for route 3, three for route 4 and two for route 5. The objective of the remaining part of this section is to show evolution of the performance and availability of the implemented system, which was calculated using the KPI Aggregated Chart grouped by the hour. Moreover,

- availability within a processing period, and
- the performance cost function

were also calculated for KPIs using a 60-min processing period. The obtained results are given in Table 5.3. As can be observed, 0.25 availability in the 5th hour is caused by the scheduled 45-min break. A similar effect is also visible in the 17h due to a 15-min break. In most cases the performance function is close to its optimal level, which is equal to 1. Note that both availability and the performance cost function can be also interpreted in percents of their maximum respective rates. This can be easily achieved by multiplying them by 100.

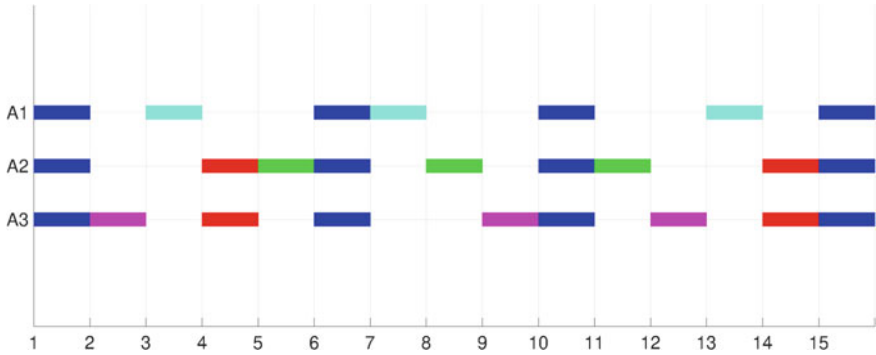


Fig. 5.4 Sample transportation request sequence

Table 5.3 Availability and performance cost function

Hour	Availability	Performance function
1	0.88	0.94
2	0.93	0.92
3	0.90	0.95
4	0.85	0.91
5	0.25	0.82
6	0.9	0.87
7	0.75	0.91
8	0.9	0.89

5.2 Monitoring the Product Rejection Rate

The objective of this section is to show a sample implementation and analysis concerning two product rejection rate monitoring schemes. In both cases, it is assumed that the number of tested products is not constant within the processing period. Let us start with the first case, which is implemented using the following environment:

KIS.Device: The operator controlling the quality of produced items is equipped with KIS.BOX, which is operating within Workspace 1.

Item rejection/acceptance: If an item is acceptable, then the operator pushes KIS.BOX Button 1 while KIS.BOX Button 2 is pushed otherwise.

Let us start with implementing the KPIs for calculating

- the total number of tested items per processing period,
- the number of rejected items per processing period.

The implementation of the first KPI boils down to counting how many times both KIS.BOX buttons were pushed, which can be performed as follows:


```
dpass=Sum[If[pass,1,0]];
dfail=Sum[If[fail,1,0]];
total=dpass+dfail;
```

where `pass` and `fail` are aliases of `button1Pressed` and `button2Pressed`, respectively, while `total` stands for the total number of tested items. The implementation of the second KPI can be performed by suitably reducing the preceding one, which yields

```
dfail=Sum[If[fail,1,0]];
```

Table 5.4 presents the obtained results concerning 20 consecutive processing periods. Note that in all cases the processing period was equal to one hour. From these results it is evident that there are in total

$$N = \sum_{i=1}^{20} n_i = 953 \quad (5.5)$$

items, among which

$$N_r = \sum_{i=1}^{20} n_{r,i} = 76 \quad (5.6)$$

are of unacceptable quality, and hence they are rejected. Thus, the ratio between the rejected and the total number of items can be obtained as follows:

$$\bar{p} = \frac{N_r}{N} = 0.0797. \quad (5.7)$$

This means that the rejection rate is around 8%.

The objective of the subsequent deliberations is to develop the p chart (cf. Sect. 4.4.2), which can be used for statistical process control pertaining to the quality of the manufactured items. Let us recall that the p chart is designed according to the following principle:

Center line: \bar{p} ,

Upper control limit (UCL):

$$\text{UCL} = \bar{p} + 3\sigma_p = \bar{p} + 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}, \quad (5.8)$$

Lower control limit (LCL):

$$\text{LCL} = \bar{p} - 3\sigma_p = \bar{p} - 3\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}. \quad (5.9)$$

Table 5.4 Quality test results

<i>i</i> -th Processing period	Rejected items ($n_{r,i}$)	Number of items (n_i)
1	4	40
2	3	47
3	3	42
4	4	48
5	4	51
6	5	41
7	4	54
8	4	51
9	5	45
10	4	45
11	3	50
12	3	55
13	2	50
14	4	52
15	3	47
16	3	39
17	5	55
18	4	50
19	4	40
20	5	51

The main assumption concerning the above principle is that the total number of samples is constant in each (*i*-th) processing period. Unfortunately, due to manual testing, such an assumption is too restrictive. Thus, one way out of this problem is to assume an average number of items per processing period. Another solution is to calculate the LCL and the UCL for each processing period separately. The KPIs calculating the UCL and the LCL for a varying *n* and \bar{p} are as follows:

UCL:

```
pbar=0.0797;
dpass=Sum[If[pass,1,0]];
dfail=Sum[If[fail,1,0]];
n=dpass+dfail;
sigmap=Power[pbar*(1-pbar)/n,0.5];
UCL=pabr+3*sigmap;
```

LCL:

```
pbar=0.0797;
```

Table 5.5 Quality test results

<i>i</i> th Processing period	Rejected items	Number of items	<i>p</i>	UCL	LCL
1	4	47	0.0851	0.1982	0
2	3	32	0.0938	0.2233	0
3	3	48	0.0625	0.1970	0
4	7	57	0.1228	0.1873	0
5	3	43	0.0698	0.2036	0
6	5	46	0.1087	0.1995	0

```

dpass=Sum[If[pass,1,0]];
dfail=Sum[If[fail,1,0]];
n=dpass+dfail;
sigmap=Power[pbar*(1-pbar)/n,0.5];
LCL=Max[pabr-3*sigmap,0];

```

Note that the LCL cannot be lower than zero, and hence there is a need for using the `Max[]` function. Finally, the center line (CL) is simply given by (5.7), which implies the KPI below:

```

pbar=0.0797;
CL=Max[pbar,pbar];

```

while the monitored rejection rate p should be calculated using the following KPI:

```

dpass=Sum[If[pass,1,0]];
dfail=Sum[If[fail,1,0]];
n=dpass+dfail;
p=dfail/n;

```

To illustrate the UCL and LCL calculation according to (5.8)–(5.9), let us consider a sample processing period in which $n = 30$. Thus, Eqs. (5.8)–(5.9) yield $UCL = 0.228$ while $LCL = \max(-0.069, 0) = 0$. Indeed, the rejection rate cannot be negative, and hence the LCL is set to zero.

Finally, using a set of four developed KPIs, one can design the p chart. Thus, according to the approach presented in Sect. 4.4.2, the above KPIs should be suitably associated with the KPI Aggregated Chart. The obtained results are presented in Fig. 5.5. For better illustration, the results are also gathered in Table 5.5. As can be observed, for the six processing periods being presented the ratio p oscillates around the center line and does not exceed the control limits.

Let us proceed to the second case, i.e., an automatic quality control system. It uses KIS.LIGHT as a communication means between an automatic quality control system and KIS.MANAGER. In particular, KIS.LIGHT Input 1 receives a false–true–false

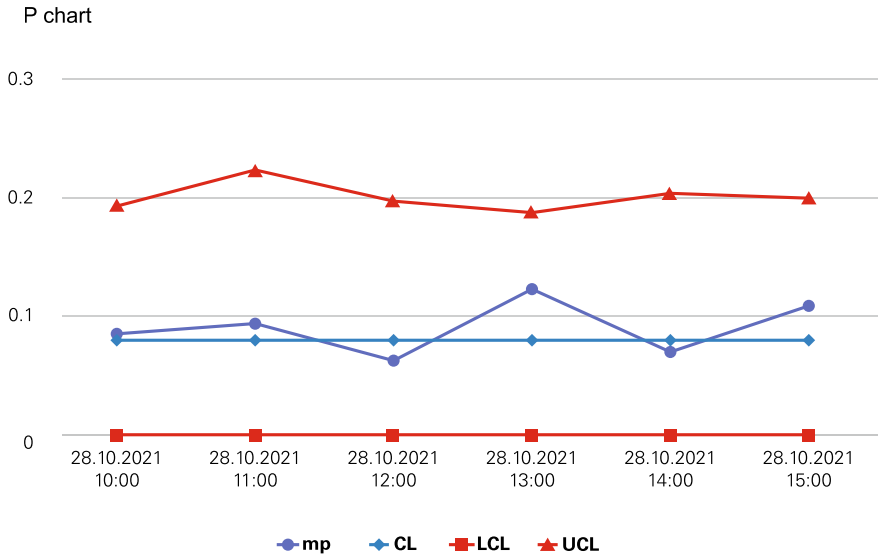


Fig. 5.5 Chart *p* for manual quality control

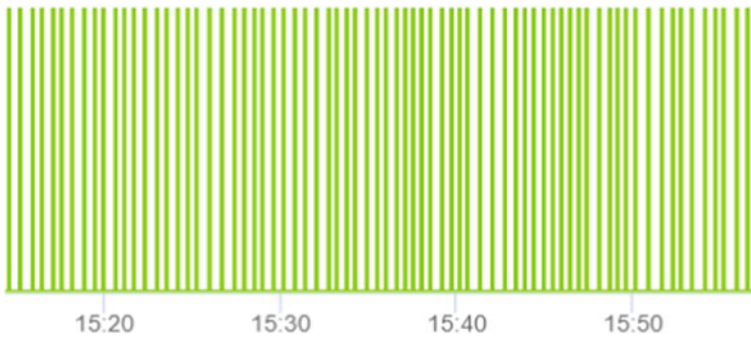


Fig. 5.6 KIS.LIGHT Input 1

sequence when the controlled item satisfies the quality requirement. Otherwise, a false–true–false sequence is sent to KIS.LIGHT Input 2. Figure 5.6 presents a sample KIS.LIGHT Input 1 sequence forming the basis for calculating the number of items which pass the quality test. As in the manual case, let us start with implementing the KPIs for calculating

- the total number of tested items per processing period,
- the number of rejected items per processing period.

Let us proceed to the implementation of the first KPI, which can be realized using the `RisingEdge` command. It counts the number of false–true sequences, and hence the KPI boils down to

```
dpass=RisingEdge[pass];
dfail=RisingEdge[fail];
total=dpass+dfail;
```

where `pass` and `fail` are aliases of `input1Status` and `input2Status` while, respectively, `total` stands for the total number of tested items. As previously, the number of items which do not satisfy the quality test can be easily obtained with the KPI as follows:

```
dfail=RisingEdge[fail];
```

Thus, the rejection rate p can be found:

```
dpass=RisingEdge[pass];
dfail=RisingEdge[fail];
n=dpass+dfail;
p=dfail/n;
```

Similarly as in the manual case, after 20 processing periods, the rejection rate was calculated as $\bar{p} = 0.098$. This means that the center line can be found as in the manual case, but $\bar{p} = 0.098$ should be used instead. Finally, the control limits are determined using KPIs implemented with

UCL:

```
pbar=0.098;
dpass=RisingEdge[pass];
dfail=RisingEdge[fail];
n=dpass+dfail;
sigmap=Power[pbar*(1-pbar)/n,0.5];
UCL=pabr+3*sigmap;
```

LCL:

```
pbar=0.098;
dpass=RisingEdge[pass];
dfail=RisingEdge[fail];
n=dpass+dfail;
sigmap=Power[pbar*(1-pbar)/n,0.5];
LCL=Max[pabr-3*sigmap,0];
```

Note that the presentation of the obtained results looks similar to that for the manual case, and hence it is omitted.

5.3 Demerit System Control

The objective of the previous section was to describe a quality control system which can be used for binary decisions concerning product quality, which can be either defective or non-defective. In the quality control nomenclature [1, 2], a product is perceived as a nonconforming one if it has one or more defects. In the case of complex products, several different kinds of defects may occur. It is, of course, evident that they are not equally important and serious. Thus, a suitable classification method is needed, which can handle severity of defects and weight them in a reasonable way. A demerit system [1, 2] can be a good remedy for such a situation. Traditionally, in such a system, there are four classes of defects:

Class A defect—very serious, due to which the product

- is not suitable for use;
- can fail in service and cannot be easily repaired;
- may cause a personal injury or a property damage.

Class B defect—serious, due to which the product

- will probably cause a Class A operating failure;
- will certainly cause less serious operating problems than the Class A one;
- will surely cause increased maintenance or a decreased lifetime.

Class C defect—moderately serious:

- will probably fail in service;
- may cause a problem less serious than a failure;
- will possibly have a reduced lifetime or increased maintenance costs;
- has a major defect in the finish, appearance or working quality.

Class D defect—minor, due to which the product

- will not fail in service;
- has a minor defect in the finish, appearance or working quality.

Let n_A , n_B , n_C and n_D represent respectively the number of Class A, B, C, D defects within the sample of n products or units. The crucial assumption is that each class of defects is independent and obeys Poisson distribution [1, 2]. The expected number of defects of each class is expressed by $n\mu_A$, $n\mu_B$, $n\mu_C$ and $n\mu_D$, where μ_i denote the expected defect per unit. Thus, the number of demerits is defined as

$$d = p_A n_A + p_B n_B + p_C n_C + p_D n_D, \quad (5.10)$$

where $p_i > 0$ stand for the class weight. A commonly used approach for selecting these weight is $p_A = 100$, $p_B = 50$, $p_C = 10$ and $p_D = 1$. Note that these parameters can be problem-specific, and hence they can be modified. The control limits for (5.10) can be calculated as follows [1, 2]:

Center line (CL):

$$CL = n(p_a\mu_a + p_b\mu_b + p_c\mu_c + p_d\mu_d), \quad (5.11)$$

Lower control limit (LCL):

$$LCL = CL - 3\sigma_d, \quad (5.12)$$

Upper control limit (UCL):

$$UCL = CL + 3\sigma_d, \quad (5.13)$$

where

$$\sigma_d = \sqrt{n(p_a^2\mu_a + p_b^2\mu_b + p_c^2\mu_c + p_d^2\mu_d)}. \quad (5.14)$$

Calculation of sample demerit control limits

Let us consider a sample demerit control system with the parameters given in Table 5.6, along with a sample of $n = 200$ units. Thus, according to (5.12),

$$CL = 200(100 \times 0.001 + 50 \times 0.0019 + 10 \times 0.0194 + 1 \times 0.01) = 79.8,$$

while (5.14) yields

$$\sigma_d = \sqrt{200(100^2 \times 0.001 + 50^2 \times 0.0019 + 10^2 \times 0.0194 + 1 \times 0.01)} = 57.793.$$

Finally, the LCL (5.12) and the UCL (5.13) are

$$LCL = 79.8 - 3 \times 57.793 = -93.579,$$

$$UCL = 79.8 + 3 \times 57.793 = 253.179.$$

Note that the negative LCL should be replaced with $LCL = 0$.

Table 5.6 Demerit control system parameters

Class	p_i	μ_i
A	100	0.001
B	50	0.0019
C	10	0.0194
D	1	0.01

> Significance of defect per unit

The above example forms the basis for developing a chart for assuring the control of a given standard, which is shaped by μ_i . This represents the expected quality level, which should take into account the economic balance between service requirements and production costs. As a result, two unappealing situations can be distinguished:

- The quality is permanently over the standard, and hence it is probable that the production is too expensive.
- The quality is permanently under the standard, and hence the cost of service/maintenance could be high. This implies the need for additional economic efforts for increasing the quality.

Under the above preliminaries, let us proceed to the KIS.ME-based implementation. As in the previous section, it is possible to develop either a manual or an automatic demerit quality control system. However, the discussion is limited to the manual case, which employs three KIS.BOXes, i.e., KIS.BOX 1, KIS.BOX 2 and KIS.BOX 3. Once a product quality check is performed, an appropriate KIS.BOX button is pressed, which expresses the class of the product. Table 5.7 presents the association of KIS.BOXes and the defect classes. Note that most products are defect-free, and hence such a class has to be included as well. The resulting demerit system is presented in Fig. 5.7. The objective of the subsequent part of this section is to provide KPIs capable of calculating (5.10)–(5.13). However, KPIs can be implemented for a single KIS.Device exclusively. Thus, it is proposed to use appropriate rules, which will be employed to feed the information about the pressed button to a single KIS.BOX, i.e., KIS.BOX 3. Indeed, as can be observed in Table 5.7, KIS.BOX 3 Button 2 is not used, and hence it will be employed for the communication purpose. A complete set of communication rules is given in Table 5.8. Figure 5.8 presents an implementation of the first rule of Table 5.8. Having such a set of rules, it is possible to proceed to the KPI implementation. Let us start with supplementary KPIs, which can be used for calculating the number of products belonging to the classes presented in Table 5.7:

Class A:

$$nA = \text{Sum} [\text{If} [x == 5, 1, 0]] ;$$

Class B:

$$nB = \text{Sum} [\text{If} [x == 4, 1, 0]] ;$$

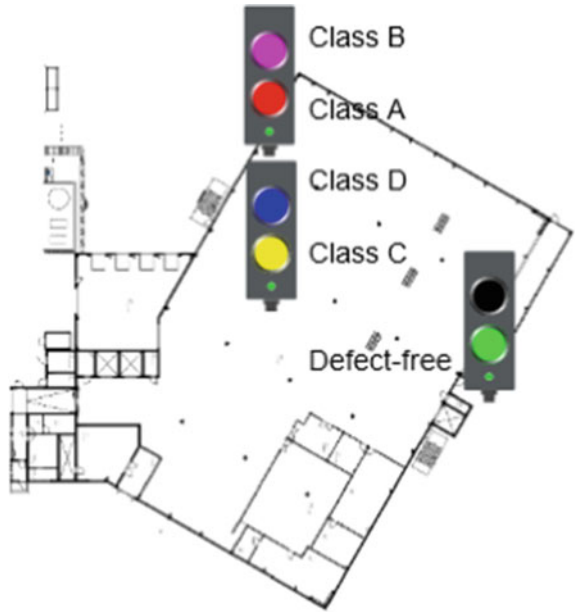
Class C:

$$nC = \text{Sum} [\text{If} [x == 7, 1, 0]] ;$$

Table 5.7 KIS.BOX-based demerit quality control

Class	KIS.BOX	Button	Color	Color no.
A	2	1	Red	5
B	2	2	Magenta	4
C	1	1	Yellow	7
D	1	2	Blue	0
Defect-free	3	1	Green	3

Fig. 5.7 Floorplan of the KIS.BOX-based demerit system



Class D:

$$nD = \text{Sum}[\text{If}[x==0, 1, 0]];$$

Class defect-free:

$$n\text{free} = \text{Sum}[\text{If}[y==3, 1, 0]];$$

where x and y are aliases of `button1ColorKPI` and `button2ColorKPI` Datapoints of KIS.BOX 3. For the implementation of (5.10)–(5.13), sample quality parameters are presented in Table 5.6. Finally, the center line and control limits of the demerit chart are given by the following KPIs:


Table 5.8 Demerit system rule base


Trigger	Actions
KB 2 Button 1 pressed	KB 3 Button 2 red
	KB 3 Button 2 black
KB 2 Button 2 pressed	KB 3 Button 2 magenta
	KB 3 Button 2 black
KB 1 Button 1 pressed	KB 3 Button 2 yellow
	KB 3 Button 2 black
KB 1 Button 2 pressed	KB 3 Button 2 blue
	KB 3 Button 2 black

Trigger ⓘ



Actions ⓘ










Fig. 5.8 Sample rule of the demerit control system

CL:

```

pa=100;
pb=50;
pc=10;
pd=1
muA=0.001;
muB=0.0019;
muC=0.0194;
muD=0.01;
nA=Sum [ If [ x==5, 1, 0 ] ];
nB=Sum [ If [ x==4, 1, 0 ] ];
nC=Sum [ If [ x==7, 1, 0 ] ];
nD=Sum [ If [ x==0, 1, 0 ] ];
nfree=Sum [ If [ y==3, 1, 0 ] ];
n=nA+nB+nC+nD+nfree;
    
```

```
CL=n*(pa*muA+pb*muB+pc*muC+pd*muD);
```

LCL:

```
pa=100;
pb=50;
pc=10;
pd=1
muA=0.001;
muB=0.0019;
muC=0.0194;
muD=0.01;
nA=Sum[If[x==5,1,0]];
nB=Sum[If[x==4,1,0]];
nC=Sum[If[x==7,1,0]];
nD=Sum[If[x==0,1,0]];
nfree=Sum[If[y==3,1,0]];
n=nA+nB+nC+nD+nfree;
CL=n*(pa*muA+pb*muB+pc*muC+pd*muD);
sd=n*(Power[pa,2]*muA+Power[pb,2]*muB+
Power[pc,2]*muC+Power[pd,2]*muD);
sigmad=Power(sd,0.5);
LCL=Max[CL-3*sigmad,0];
```

UCL:

```
pa=100;
pb=50;
pc=10;
pd=1
muA=0.001;
muB=0.0019;
muC=0.0194;
muD=0.01;
nA=Sum[If[x==5,1,0]];
nB=Sum[If[x==4,1,0]];
nC=Sum[If[x==7,1,0]];
nD=Sum[If[x==0,1,0]];
nfree=Sum[If[y==3,1,0]];
n=nA+nB+nC+nD+nfree;
CL=n*(pa*muA+pb*muB+pc*muC+pd*muD);
sd=n*(Power[pa,2]*muA+Power[pb,2]*muB+
Power[pc,2]*muC+Power[pd,2]*muD);
sigmad=Power(sd,0.5);
LCL=CL+3*sigmad;
```

Table 5.9 KPI-based calculation of quality control results

<i>i</i> -th Processing period	n_a	n_b	n_c	n_d	Defect-free	n
1	0	2	2	1	185	190
2	0	0	3	3	224	230
3	0	0	7	0	193	200
4	0	1	2	2	235	240
5	0	1	6	1	191	199
6	0	0	5	3	178	186

Table 5.10 Numerical data of the demerit chart

<i>i</i> -th Processing period	d	CL	LCL	UCL
1	121	75.8100	0	244.7982
2	33	91.7700	0	277.6974
3	70	79.8000	0	253.1782
4	72	95.7600	0	285.6863
5	111	79.4010	0	252.3452
6	53	74.2140	0	241.4139

The developed demerit system was validated using a one-hour processing period. First, the supplementary KPIs were employed to determine the number of products belonging to the classes defined in Table 5.7. The obtained results are presented in Table 5.9. Let us proceed to the results concerning the demerit chart, which are shaped by the monitored demerit number d . It should evolve around the center line (CL) and should be bounded by the control limits. The obtained results are presented in Table 5.10 and visualized using the KPI Aggregated Chart, which is portrayed in Fig. 5.9. The results clearly indicate that the monitored process is in the state of control.

5.4 Overall Equipment Effectiveness

The main objective of this section is to show how to use KIS.ME for calculating the performance of given equipment. The discussion starts with recalling the concept of overall equipment effectiveness (OEE) [3, 4], which can be perceived as a key measurement tool for assessing both productivity and efficiency. As indicated in [4],

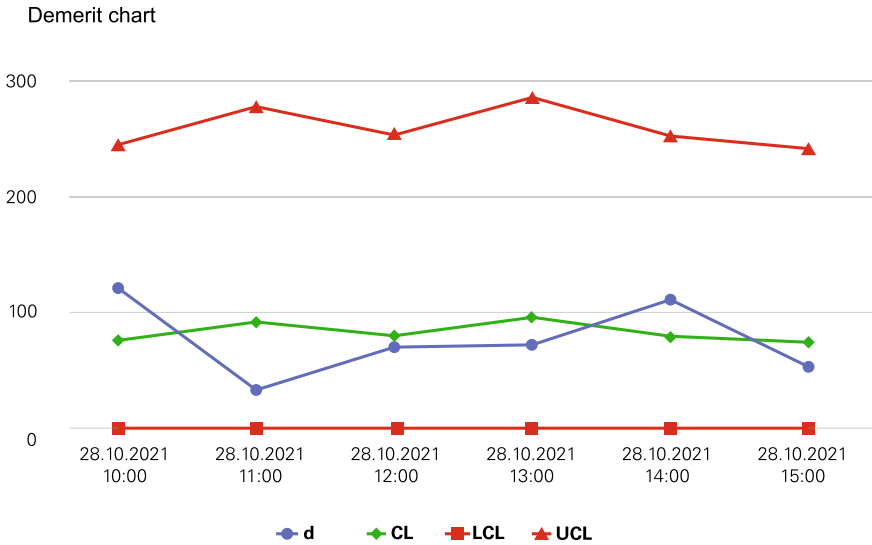


Fig. 5.9 Demerit chart for manual quality control

OEE is a hierarchy of measures that exhibit how efficiently a manufacturing operation is performed. This indicator is stated in a very general form, and hence it makes it possible to perform an efficient comparison between manufacturing units in different departments, organizations, etc. The core features of OEE are as follows [3, 4]:

- identification of equipment potential;
- identification and tracking of the losses;
- identification of opportunities for increasing equipment performance.

As a result, OEE can be used for

- increasing productivity,
- decreasing the overall cost,
- increasing the awareness about equipment productivity,
- extending the equipment operational life time.

The crucial components of OEE are the following: availability:

$$A = \frac{t_P - t_S}{t_P} = \frac{t_R}{t_P}, \quad (5.15)$$

where t_P is a planned production time, t_S stands for the unplanned stop or downtime, while t_R signifies the run time.

Performance:

$$P = \frac{t_i n_p}{t_P - t_S} = \frac{t_i n_p}{t_R}, \tag{5.16}$$

where t_i is the ideal single part manufacturing time and n_p stands for the total number of manufactured parts, i.e., both defective and defect-free ones. Quality:

$$Q = \frac{n_p - n_d}{n_p} = \frac{n_g}{n_p}, \tag{5.17}$$

where n_d and n_g stand for the number of defective and defect-free parts, respectively.

Finally, the OEE indicator is simply given by

$$OEE = A \times P \times Q. \tag{5.18}$$

The objective of the remaining part of this section is to show how to employ KIS.ME for calculating OEE. It can also be expressed in percents, which can be easily attained by multiplying (5.15)–(5.17) by 100. Note that a world class value of OEE should be at the level of 85% or higher. Let us also note that the calculation of planned production time obeys

$$t_P = t_A - t_B, \tag{5.19}$$

where t_A and t_B stand for the available and planned break times.

Sample OEE calculation

Let us consider an example equipment, which is characterized by the parameters given in Table 5.11. According to (5.19), the planned production time $t_P = t_A - t_B = 390$ [min]. This implies that $A = \frac{t_P - t_S}{t_P} = \frac{350}{390} = 0.897$ or 89.7%. Subsequently, the performance can be calculated with (5.16), which is equal to $P = 0.952$ or 95.2%. The quality is obtained with (5.17), which is equal to $Q = 0.99$ or 99%, equivalently. Finally, OEE can be calculated according to (5.18), which gives $OEE = 0.845$ or 84.5%.

Table 5.11 Sample production equipment parameters

Parameter	Value
t_A	420 (min)
t_B	30 (min)
t_S	40 (min)
t_i	$\frac{1}{60}$ (min)
n_p	20000 (items)
n_d	200 (items)

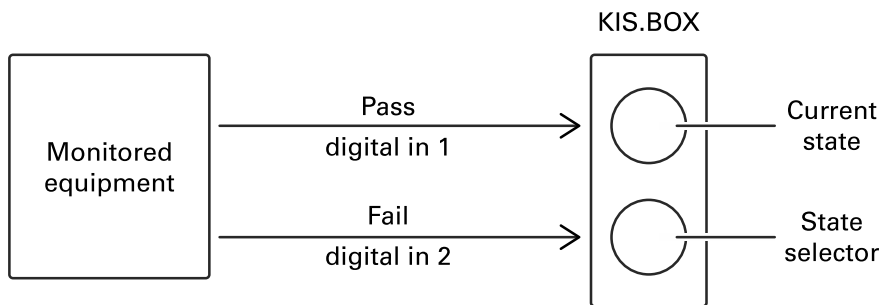


Fig. 5.10 KIS.ME-based OEE data gathering scheme

Before proceeding to the OEE implementation, a KIS.ME infrastructure has to be defined. In particular, it should allow identification and measurement of

- the unplanned downtime t_S and its cause,
- the total number of manufactured parts n_p ,
- the number of defective parts n_d .

The employed infrastructure is portrayed in Fig. 5.10. As can be observed, the system has automatic quality control, which is connected with KIS.BOX digital inputs. The quality control system is actually the same as the one presented in Sect. 5.2, but KIS.BOX is used instead of KIS.LIGHT. Indeed, it employs KIS.BOX as a communication means between an automatic quality control system and KIS.MANAGER. In particular, KIS.BOX Input 1 receives a false–true–false sequence when the controlled item satisfies the quality requirement. Otherwise, a false–true–false sequence is sent to KIS.BOX Input 2. A sample sequence is presented in Fig. 5.6. Subsequently, note that the KIS.BOX Button 2 operational LED color is initially set to green using the KIS.BOX digital twin (cf. Sect. 2.6). This is a normal operation of the equipment.

Let us proceed to the identification of the cause of downtime. First, it is assumed that the equipment operator makes a decision about the current downtime state of the equipment, which may exhibit one of the modes given in Table 5.12. In particular, the state-space model concept (cf. Sect. 2.10) is utilized to change the color of the KIS.BOX Button 1 operational LED according to Table 5.12. This operation depends on the trigger which is related to pressing KIB.BOX Button 1. Once an appropriate color is selected, the equipment operator acknowledges the current state by pressing KIS.BOX Button 2. As a result, its operational LED color is changed into that of the KIS.BOX Button 1 operational LED. Thus, any alteration of the equipment state should be realized in the same way. Note also that one can freely modify or extend the states proposed in Table 5.12.

Let us proceed to the KPI implementation. First, availability has to be calculated according to (5.15). For that purpose it is necessary to assume that the planned production time t_P is given. Thus, let $t_A = 420(\text{min})$ and $t_p = 390(\text{min})$. As a result, the KPI calculating availability within a selected processing period is given by

Table 5.12 Run and downtime states

State	Color	Color no.
Run	Green	3
Equipment breakdown	Red	5
Setup and adjustment	Blue	0
Minor breakdowns	Yellow	7
Planned break	Magenta	4

```

tp=390;
tr=Sum[If[x == 3,Duration[x],0]]/60000;
A=tr/tp;
    
```

where x is an alias of the `button1ColorKpiDuration` Datapoint. Finally, to assess the quality, the KPI calculation results should be aggregated with KPI charts using the SUM aggregation method (see, e.g., Sect. 4.3.2) and the t_A aggregation period. Using a similar way of implementation, one can formulate KPIs calculating the sums of downtimes:

```

Equipment breakdown: dred=Sum[If[x == 5,Duration[x],0]]/60000;
Setup and adjustment: dbblue=Sum[If[x == 0,Duration[x],0]]/60000;
Minor breakdowns: dyellow=Sum[If[x == 7,Duration[x],0]]/60000;
    
```

as well as the occurrence number of such states:

```

Equipment breakdown: nred=Sum[If[x == 5,1,0]];
Setup and adjustment: nblue=Sum[If[x == 0,1,0]];
Minor breakdowns: nyellow=Sum[If[x == 7,1,0]];
    
```

Let us proceed to performance calculation (5.16). The KPI calculating local performance within the j -th processing period, i.e.,

$$P_j = \frac{t_i n_{p,j}}{t_{R,j}}, \tag{5.20}$$

can be derived using (with $t_i = 1[\text{min}]$)

```

ti=1;
ng=RisingEdge[y];
nd=RisingEdge[z];
np=nd+ng;
tr=Sum[If[x == 3,Duration[x],0]]/60000;
Pj=ti*np/tr;
    
```

where x , y and z are aliases of `button1ColorKpiDuration`, `input1Status` and `input2Status`, respectively. Having k processing periods, one can aggregate

the results of the above KPI. Unfortunately, there is no aggregation which makes it possible to determine total performance (5.16). A good remedy to such a problem is to define two separated KPIs:

Ideal manufacturing time of the n_p parts:

```
ti=1;
ng=RisingEdge[y];
nd=RisingEdge[z];
np=nd+ng;
tidea=ti*np;
```

Run time t_R :

```
tr=Sum[If[x == 3,Duration[x],0]]/60000;
```

and observe their relation using the KPI Pie Chart with the SUM aggregation method. Similar issues are encountered while calculating quality with (5.17). Indeed, local quality within the j -th processing period, i.e.,

$$Q_j = \frac{n_{g,j}}{n_{p,j}}, \quad (5.21)$$

can be calculated with

```
ng=RisingEdge[y];
nd=RisingEdge[z];
np=nd+ng;
Qj=ng/np;
```

Similarly, as the global quality (5.17) cannot be directly calculated, a good remedy is to define two separate KPIs:

Number of defect-free parts n_g : `ng=RisingEdge[y]` ;
 Number of parts n_p :

```
ng=RisingEdge[y];
nd=RisingEdge[z];
np=nd+ng;
```

and observe their relation using the KPI Pie Chart with the SUM aggregation method. Finally, the determination of OEE can be realized directly. Indeed, by substituting (5.15)–(5.17) to (5.18), one can observe that

$$OEE = \frac{t_i n_g}{t_p}, \quad (5.22)$$

which can be directly obtained with the following KPI:

```
ti=1;
tp=390;
ng=RisingEdge[y];
OEE=ti*ng/tp;
```

Finally, OEE can be visualized and aggregated (with the SUM method) using a selected KPI chart.

5.5 Training Exercises

5.1 Small transportation system

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Let us consider two transportation routes:

Route red:

$$R_r = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], \quad (5.23)$$

Route green:

$$R_g = [A, B, C, D, E, F, G, H, I, J]. \quad (5.24)$$

2. Each route is going through 10 points, which can be perceived as the virtual supermarkets and assembly stations.
3. Each virtual transportation task is realized in the following way:
 - Step 0: set $i = 1$;
 - Step 1: write the i -th name of the transportation point on a paper sheet (e.g., A);
 - Step 2: perform virtual transportation by waiting a suitable period of time;
 - Step 3: set $i = i + 1$. If $i > 10$, then STOP, else go to Step 1.
4. The ideal transportation times for the first and second route are $t_r = \frac{11}{60}$ (min) and $t_g = \frac{13}{60}$ (min).
5. Using the approach presented in Sect. 5.1, implement two rules for acknowledging the transportation tasks (cf. Fig. 5.1).
6. Implement the KPI counting the number of transportation tasks n_r realized through R_r .
7. Implement the KPI counting the number of transportation tasks n_g realized through R_g ;
8. Implement the KPI calculating the performance of the transportation system within a processing period, which is expressed by

$$J = \frac{1}{t_p} (n_r t_r + n_g t_g), \quad (5.25)$$

where t_p is the processing period. (Hint: Use `Interval[]` command with transforming its value from milliseconds to minutes);

9. Implement the KPI calculating the availability of the transportation system within a processing period:

$$A = \frac{t_r}{t_p}, \quad (5.26)$$

where t_r stands for the run time, i.e., the duration sum for which the KIS.BOX Button 2 operational LED is not black.

10. Use arbitrary KPI charts to present the obtained results of (5.25).
11. By using the KIS.BOX digital twin, i.e., by setting the color of the KIS.BOX Button 1 operational LED, repeat cyclically the following route assignment:

$$R_g, R_g, R_r, R_r, R_r, R_g, R_r, R_g, R_g \quad (5.27)$$

and realize each transportation task according to Step 3.

12. What can you say about the obtained values of (5.25) and (5.26)?

5.2 Manual quality control

Exercise requirements: The exercise requires access to one KIS.BOX.

1. Implement the manual quality control system presented in Sect. 5.2.
2. Use the Datapoint chart to visualize the system's behaviour.
3. Implement the KPI calculating the number of items which pass the quality control.
4. Implement the KPI calculating the number of items which fail the quality control.
5. Implement the KPI determining the total number of tested items.
6. Using the KPI Pie Chart, visualize the relation between the fail/pass items.
7. Perform a virtual quality control action by writing consecutive natural numbers on a sheet of paper. If a number can be divided by 10, then press KIS.BOX Button 2 (fail), else press KIS.BOX Button 2 (passed).
8. Determine the ratio between the number of rejected and all items \bar{p} .
9. Using (5.8)–(5.9), calculate the lower and upper control limits of the p chart.

5.3 Virtual production system with quality control

Exercise requirements: The exercise requires access to one KIS.LIGHT.

1. Using Rule engine, implement a state-space model (cf. Sect. 2.10) which will cyclically change the KIS.LIGHT operational LED color according to Table 5.13.
2. Using the KIS.LIGHT digital twin (cf. Sect. 2.6), initialize its operational LED color to be equivalent to the run state (green color).

Table 5.13 Run and downtime states of the virtual production system

State	Color	Color No.	State period (min)
Run	Green	3	50
Equipment breakdown	Red	5	10
Setup and adjustment	Blue	0	5
Minor breakdown	Yellow	7	2
Planned break	Magenta	4	1

3. Using the Datapoint Chart, visualize the performance of the system.
4. Using Rule engine, implement a rule mechanism which during the run state (green color) will, every minute, change KIS.LIGHT digital output 1 according to the false–true–false sequence.
5. Using the Datapoint Chart, visualize KIS.LIGHT digital output 1.
6. Using Rule engine, implement a rule mechanism which during the run state (green color) will, every minute, change KIS.LIGHT digital output 1 according to the false–true–false sequence;
7. Using the Datapoint Chart, visualize KIS.LIGHT digital output 1.
8. Using Rule engine, implement a rule mechanism which during the run state (green color) will, every 45 min, change KIS.LIGHT digital output 2 according to the false–true–false sequence.
9. Using the Datapoint Chart, visualize KIS.LIGHT digital output 2.
10. let us imagine that each false–true–false sequence on KIS.LIGHT digital input 1 corresponds to a good item while the same sequence on KIS.LIGHT digital input 2 signifies a failed one. What can you say about FPY? What is the expected value of FPY?

5.4 Overall equipment efficiency of the virtual production system

Exercise requirements: The exercise requires access to one KIS.LIGHT and completion of Exc. 5.3.

1. Implement the p chart for the quality control system;
2. Select the production system available time t_A , and use Table 5.13 to calculate the overall planned break time t_B within t_A .
3. Calculate the planned production time t_P .
4. Select a uniform processing period for all KPIs, e.g., one hour, implemented in this exercise.
5. Implement KPIs calculating the sum of durations corresponding to all individual states given in Table 5.13.
6. Visualize the KPI results obtained in the preceding point using the KPI Pie Chart with the SUM aggregation method.
7. Implement KPIs calculating the occurrence number of individual downtime states, i.e., equipment breakdown, setup and adjustment, minor breakdowns.

8. Visualize the KPI results obtained in the preceding point using the KPI Single Period Chart with the SUM aggregation method.
9. Implement the KPI calculating the availability of the system (5.15), and visualize the obtained results using the KPI Pie Chart with the SUM aggregation method.
10. Implement two KPIs calculating the number of defective n_d and defect-free n_g items.
11. Visualize the KPI results obtained in the preceding point using the KPI Pie Chart with the SUM aggregation method.
12. Implement the KPI calculating the total number of manufacture items n_P and visualize the obtained results using KPI Aggregated Chart with the SUM aggregation method.
13. Analyse the results obtained in the preceding step and determine an ideal (almost impossible to achieve) manufacturing time of a single time t_i .
14. Implement two KPIs calculating an ideal manufacturing time of n_P items and the actual run time t_R .
15. Visualize the KPI results obtained in the preceding point using the KPI Pie Chart with the SUM aggregation method.
16. Implement the KPI calculating the overall equipment efficiency (5.18) and visualize the obtained results using the KPI Single Period Chart.

5.6 Concluding Remarks

The main objective of this chapter was to utilize the methods and tools described in the preceding parts for designing a practical set of process monitoring and control schemes. All of them are relatively easy to implement and enable intuitive control of the monitored system. The chapter opened with a transportation system that operates on a set of routes. First, it was shown how to communicate the desired transportation actions between KIS.MANAGER and transporter operators. The second objective was to develop suitable measures for assessing the performance of the transportation system. For that purpose, the concept of an ideal rout time was introduced, which forms the basis for the performance cost function. Thus, with appropriate control of the transportation system, one can optimize this function. Additionally, the proposed function is very easy to interpret as its value for the optimal control is equal to 1, which can be perceived as 100% performance. The proposed approach also allows determining the availability of the transportation system, which can be used as an additional measure for performance improvements. The second process which was introduced in this chapter pertains to a quality control system, which can be either manual or automatic. Irrespective of the selected method, it was shown how to determine a set of suitable statistical measures along with the p chart. Such a quality control system is capable of making binary quality decisions about the product being controlled. Thus, to overcome this restriction, a demerit quality control system was introduced. It allows indicating various defect classes, and hence, instead of controlling the rejecting rate, it is proposed to monitor the so-called demerit number. For

that purpose, the demerit chart was developed, which provides effective measures for controlling the quality of manufactured products. The last process monitoring strategy aimed at calculating and visualizing overall equipment efficiency, which is widely perceived as a key measurement tool for assessing both productivity and efficiency. In particular, it was shown how to efficiently observe availability, performance, and quality of a given manufacturing equipment. Finally, the chapter was summarized with a set of training exercises, which can be considered the master level test concerning KIS.ME-oriented skills.

References

1. L.A. Jones, W.H. Woodall, M.D. Conerly, Exact properties of demerit control charts. *J. Qual. Tech.* **31**(2), 207–216 (1999)
2. D.C. Montgomery, *Introduction to Statistical Quality Control* (Wiley, London, 2020)
3. R.C. Hansen, *Overall Equipment Effectiveness: A Powerful Production/Maintenance Tool for Increased Profits* (Industrial Press Inc., New York, 2001)
4. D.H. Stamatis, *The OEE Primer: Understanding Overall Equipment Effectiveness, Reliability, and Maintainability* (CRC Press, Boca Raton, 2017)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 6

Towards Advanced Applications



6.1 Modelling Users and Their Interactions

Unlike automated assembly systems, human operated manufacturing and assembly ones encounter various unappealing effects associated with human behaviour uncertainty. The objective of this section is to provide two examples pertaining to modeling human users and their interactions [1] with the fuzzy logic approach [2–4]. The fuzzy logic paradigm seems to be a natural modelling tool for such a task as its origins stem from human inference behaviour. Indeed, fuzzy logic has been used for modelling human reliability in the process industry [5–7] as well as human error analysis [8]. There are also works utilising fuzzy logic for human factor modelling in preventive maintenance actions [9] and estimating a context-specific human error rate [10].

6.1.1 Assembly Process

Let us proceed to the first case, i.e., modelling human behaviour in the assembly process with fuzzy logic. For that purpose a pair of KIS.Devices is employed, i.e., KIS.BOX and KIS.LIGHT. For the purpose of illustration, let us consider a sample manual assembly process, which involves two phases [11] (see Fig. 6.1):

1. battery cell mounting,
2. cell-controller linking.

A general overview of the manual assembly station with the KIS.ME infrastructure is presented in Fig. 6.2. Let us recall that KIS.LIGHT possesses two digital inputs, which can be connected with two photoelectric sensors (see Sect. 3.1 for an illustrative example). Let us imagine that there are transportation means, e.g., conveyor belts, which provide battery cells and cell controllers. Table 6.1 presents all possible situations related to this. Using KIS.MANAGER Rule engine (cf. Sect. 2.10), one can implement rules governing such behaviour of the KIS.LIGHT operational LED

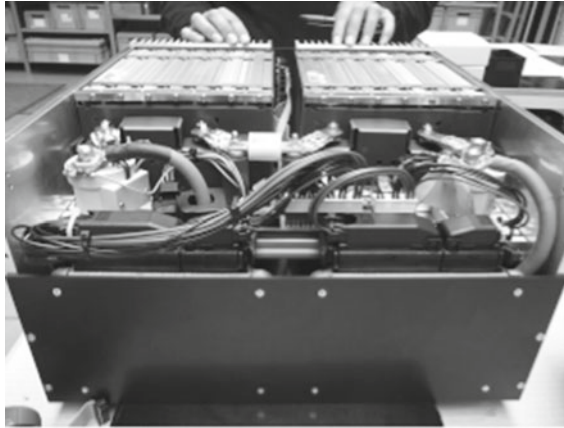


Fig. 6.1 Manual battery assembly

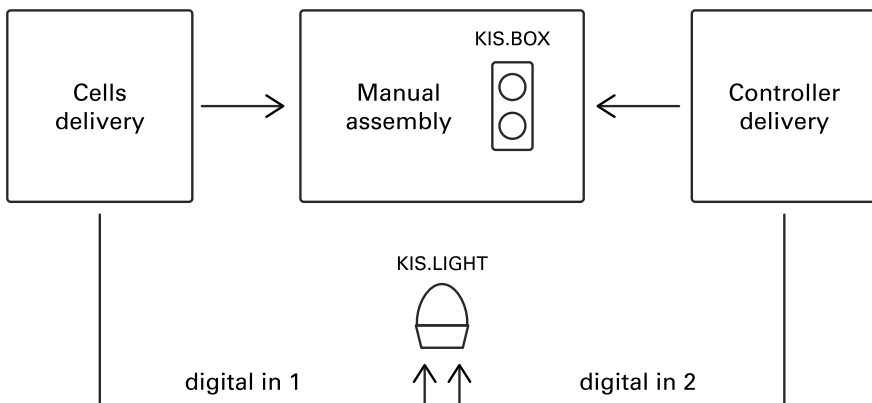


Fig. 6.2 General overview of the manual assembly station with the KIS.ME infrastructure

depending on its digital inputs. As a result, if its color is red, then there are no components. If one of them is available, then the color is green. Finally, if both of them are available, then the color is blue and it alarms the worker that the assembly process can be started. Thus, a necessary condition for mounting the battery is that all components be available. Let us proceed to describe all possible working states, which are listed in Table 6.2. Similarly as in the KIS.LIGHT case, the transition between States 1–5 can be realized using Rule engine. Moreover, as a transition trigger, one can use a KIS.BOX Button 1 pressing action. This action ends the deployment of KIS.Devices to the above assembly station. Note that the structure presented in Fig. 6.2 is very general, and hence it can be tailored to a wide range of assembly tasks. Having the KIS.Device infrastructure, it is possible to monitor the performance of the human operated assembly system using visualization, SPC and OEE strategies, described in

Table 6.1 KIS.LIGHT states

KL operational LED color	KL digital input 1	KL digital input 2
Red	0	0
Green	1	0
Green	0	1
Blue	1	1

Table 6.2 KIS.BOX states

State	KB Button 1 operational LED	KB Button 2 operational LED	Action
1	Blue	Blue	Can start
2	Red	Blue	Cell mounting
3	Green	Blue	Cell mounting completed
4	Green	Red	Cell-controller linking
5	Green	Green	Mounting completed

Chaps. 4–5. This is, however, beyond the scope of this section. Indeed, the problem is to obtain a time-driven model of assembly operations, which will include information about human experience and performance related with the working time within the shift.

For that purpose, let us recall that the time evolution of all states can be easily measured within KIS.MANAGER using the Data trend chart widget (cf. Sect. 2.7), which allows real-time analysis of operator performance. The historical data can easily be saved to a CSV file, and hence it can be further processed in external software. Thus, the objective is to model human behaviour taking into account the following issues:

- varying human experience and performance,
- a variety of batteries to be mounted.

Thus, the crucial problem is to model the realization times of States 2 and 4 in Table 6.2. For that purpose, two general models are introduced:

$$c_m = f_m(n_c, C_c, m_p, e_x, t_s), \tag{6.1}$$

$$c_l = f_l(n_c, e_x, t_s), \tag{6.2}$$

where

- c_m is the cell mounting time;
- c_l signifies the controller linking time;
- $f(\cdot)$ is the model structure;
- e_x represents the experience of an operator, e.g., in the range between 0 and 10;

Table 6.3 Fuzzy premise variables

Variable	Description	Intervals
e_x	Advanced	6–10
	Intermediate	3–7
	Beginner	0–4
t_s	Long	5–8
	Medium	3–6
	Short	0–4

- t_s is the working time within the shift, e.g., from 0 to 8 h;
- n_c stands for the number of cells to be mounted inside the battery pack;
- m_p is the mass of the battery pack.

Having all necessary ingredients, it is possible to provide the structure of (6.1)–(6.2). For that purpose, it is proposed to use the Takagi–Sugeno (TS) fuzzy logic approach [12, 13]. TS models are widely used for various modelling tasks (see [13] and the references therein). Their attractiveness is especially important in the case considered since there are two linguistic variables, presented in Table 6.3. They can be directly obtained, and hence they can form the so-called premise variables [13] for the TS counterparts of (6.1)–(6.2). According to the above KIS.Device infrastructure, each worker is associated with KIS.BOX, and hence its experience e_x can be directly obtained. As can be observed in Table 6.3, fuzzy logic allows smooth transitions between the groups. Indeed, with a degree of membership [13], a worker can be in one group. Similarly, the worker can be in a different group with another membership degree. The same feature pertains to the working time within the shift.

Under these preliminaries, the TS model of (6.1) is as follows:

$$\text{IF } t_s \in \mathbb{T}_{s,j} \text{ and } e_x \in \mathbb{E}_{x,i}, \text{ THEN} \\ c_m = p_{1,m}^{i,j} n_c + p_{2,m}^{i,j} m_p + p_{3,m}^{i,j} C_c, \quad i, j = 1, \dots, 3, \quad (6.3)$$

where p_i are the model parameters which have to be determined while $\mathbb{E}_{x,j}$ and $\mathbb{T}_{s,i}$ are fuzzy sets associated with Table 6.3. Each set can be shaped with three membership functions, which can be, e.g., of the Gaussian form:

$$\mathcal{G}^i(e_x, m_{x,i}, s_{x,i}) = \frac{1}{\sqrt{2\pi s_{x,i}}} e^{-\frac{(e_x - m_{x,i})^2}{2s_{x,i}}}, \quad i = 1, \dots, 3, \quad (6.4)$$

$$\mathcal{G}^j(t_s, m_{s,j}, s_{s,j}) = \frac{1}{\sqrt{2\pi s_{s,j}}} e^{-\frac{(t_s - m_{s,j})^2}{2s_{s,j}}}, \quad j = 1, \dots, 3, \quad (6.5)$$

where m_x, m_s are the centres of Gaussian functions while s_x and s_s define their dispersion. Having the membership functions, let us define the normalized rule firing strength:

$$\mu_{i,j} = \frac{\mathcal{G}^i(e_x, m_{x,i}, s_{x,i})\mathcal{G}^j(t_s, m_{s,j}, s_{s,j})}{\mathcal{G}_t}, \quad i, j = 1, \dots, 3, \quad (6.6)$$

where

$$\mathcal{G}_t = \sum_{i=1}^3 \sum_{j=1}^3 \mathcal{G}^i(e_x, m_{x,i}, s_{x,i})\mathcal{G}^j(t_s, m_{s,j}, s_{s,j}). \quad (6.7)$$

Finally, the model (6.3) can be transformed into

$$c_m = \sum_{i=1}^3 \sum_{j=1}^3 \mu_{i,j} \left(p_{1,m}^{i,j} n_c + p_{2,m}^{i,j} m_p + p_{3,m}^{i,j} C_c \right). \quad (6.8)$$

As a result, the TS model can be expressed in the following form:

$$c_m = \sum_{i=1}^3 \sum_{j=1}^3 \mu_{i,j} r_m^T p_m^{i,j}, \quad (6.9)$$

where $r_m = [n_c, m_p, C_c]^T$ and $p_m^{i,j} = [p_{1,m}^{i,j}, p_{2,m}^{i,j}, p_{3,m}^{i,j}]^T$ are the regressor and parameter vectors, respectively. The final step boils down transforming (6.9) to the regressor form:

$$c_m = r_m^T p_m, \quad (6.10)$$

where

- $r_m = [\mu_{1,1} r_m^T, \mu_{1,2} r_m^T, \dots, \mu_{2,2} r_m^T]^T$,
- $p_m = [(p_m^{1,1})^T, (p_m^{1,2})^T, \dots, (p_m^{3,3})^T]^T$.

Using a similar line of reasoning, the TS model (6.2) can be derived:

$$c_l = r_l^T p_l, \quad (6.11)$$

where

- $r_l = [\mu_{1,1} r_l^T, \mu_{1,2} r_l^{1,2} T, \dots, \mu_{3,3} r_l^T]^T$,
- $p_l = [(p_l^{1,1})^T, (p_l^{1,2})^T, \dots, (p_l^{3,3})^T]^T$.

with $r_l = n_c$.

One important feature related to the models (6.10)–(6.11) is that they are linear with respect to p_m and p_l . This appealing feature makes it possible to use the celebrated least-square algorithm [12, 13] for estimating these parameters.

The objective of the remaining part of this section is to provide a concise outline of the general algorithm for designing (6.10)–(6.11):

KIS.BOX assignment: assign workers with different experience (see Table 6.3) to different KIS.BOXes.

Production plan: determine a strategy for gathering representative and possibly large data sets concerning workers with different experience that operate according to the battery production plan.

Data gathering: perform the data gathering procedure:

- obtain the data from the Data trend chart;
- synchronize the obtained data with the production plan;
- based on the KIS.BOX assignment, extend the data set with worker experience.

Model design: use the recursive least square algorithm to determine (6.10)–(6.11).

> Highlights

The model (6.10)–(6.11) may have several prospective applications. Having a manufacturing plan corresponding to the required number of batteries as well as group of workers performing within the shifts, one can use (6.10)–(6.11) to

1. schedule production and estimate its feasibility;
2. schedule the work of individual workers, i.e., the frequency of delivering components;
3. make the production process transparent, and hence increase various KPIs, e.g., OEE;
4. use it as a digital twin of the worker, which can be employed for their training and continuous skill improvements.

6.2 Transportation Process

The objective of this section is to present an approach for modelling forklift operator performance within a warehouse, which has been recently developed by the authors [1, 14]. The proposed framework is designed for a fleet of cooperating forklifts working within a high storage warehouse. A sample part of such a system is presented in Figs. 6.3–6.4. As can be observed, KIS.Devices are used as communication tools. Thus, each forklift is equipped with KIS.BOX while KIS.LIGHTs are installed on each transfer station. The transfer station is defined as a place for storing an item (product, materials, etc.), which has been delivered to it via different transportation means, e.g., automated guided vehicles [15, 16].

Let us start with stating that an item associated with the k -th transportation event is identified by

$$\mathbb{L}(k) = \{s(k), t(k), d(k), m(k), q(k)\}, \quad (6.12)$$

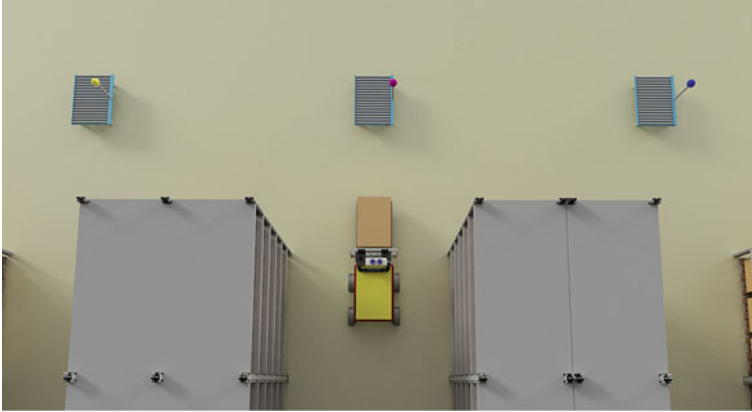


Fig. 6.3 Sample part of the warehouse shopfloor (top view)



Fig. 6.4 Sample part of the warehouse shopfloor (side view)

where

- $s(k)$ stands for the storage place identifier in a warehouse;
- $t(k)$ is the transfer station identification number;
- $d(k)$ is the distance (in meters) between the places $t(k)$ and $s(k)$;
- $m(k)$ signifies the item mass (in kg);
- $q(k)$ denotes the item volume (in m^3).

The objective of the remaining part of this chapter is to provide a concise procedure for designing a forklift driver performance model using KIS.ME. Let \mathbb{P} denote the set of all require transportation events, which is defined as follows:

$$\mathbb{P} = \{\mathbb{L}(0), \mathbb{L}(1), \dots, \mathbb{L}(n_E)\}, \tag{6.13}$$

where n_E signifies the number of transportation events.

> Realization constraints

For the sake of simplicity, the following is assumed:

- The transfer stations are situated close to the entrance of each aisle (see Fig. 6.3).
- Each forklift is operating in a single aisle.
- An item designated to a given aisle can be delivered to its direct or closest neighboring transfer stations (see the illustrative example given in Fig. 6.3).
- Each KIS.LIGHT operational LED illuminates in a different color, and hence it uniquely identifies the transfer station, i.e., the transfer station identifier $t(k)$ is equal to the numerical value of its color (see Table 2.2).
- KIS.LIGHT digital inputs uniquely define the designated aisle of an item (see Table 6.4), which can be in front of it or on the left or right hand side. It is assumed that the states of digital inputs are set by the transportation means which delivers an item to the transportation station. Note also that if an item is collected by a forklift, then the digital inputs are set to the “None” state (see Table 6.4), which signifies the fact that the transfer station is empty.
- it is assumed that there exists a transportation system which cooperates with KIS.ME and delivers items to the appropriate transfer stations;
- the availability of a forklift is signified by the facts that of the both KIS.BOX Button 1 and 2 the operational LED colors are black, i.e., they do not illuminate.
- The item can be released at a transfer station iff it is empty.

Thus, transportation from the transfer stations to the storage places is realized as follows:

- Step 0: Set $k = 1$;
- Step 1: The k -th item is delivered to $t(k)$, which is identified by $\mathbb{L}(k)$.
- Step 2: Using the $t(k)$ transfer station’s KIS.LIGHT operational LED color (numerical value $t(k)$) and digital inputs states (Table 6.4), KIS.MANGER determines the designated aisle, i.e., a forklift and its associated KIS.BOX.
- Step 3: If of the both KIS.BOX Button 1 and Button 2 the operational LED colors are black, then their color is changed to the one with the numerical value equal to $t(k)$ (blue in Fig. 6.3).
- Step 4: The forklift operator presses KIS.BOX Button 1 and its operational LED starts lighting in red. The operator starts the $\mathbb{L}(k)$ transportation event by collecting the k -th item and going to the storage place $s(k)$.

Table 6.4 Aisle assignment to the transfer station

Aisle	KIS.LIGHT digital input 1	KIS.LIGHT digital input 2
In front	1	1
Left hand side	1	0
Right hand side	0	1
None	0	0

- Step 5: The forklift operator arrives at $s(k)$, pushes KIS.BOX Button 1 again and then its operational LED changes its color to black.
- Step 6: The forklift operator presses KIS.BOX Button 2 and its operational LED starts lighting in red. The operator releases the item at $s(k)$ and moves to the beginning of the aisle.
- Step 7: The operator arrives at the beginning of the aisle, presses KIS.BOX Button 2 and its operation LED changes its color to black.
- Step 8: If $k < n_E$, then set $k = k + 1$, else STOP.

Note that if the condition of Step 3 is not satisfied, then KIS.MANGER Rule engine waits until it is feasible. Moreover, the black and red colors are used for the task identification purpose, and hence they cannot be used for the identification of the transfer stations. Finally, the objective of this data acquisition is to collect measurements associated with

- Steps 4–5: the duration $f(k)^m$ in which the operational LED color of the i -th forklift KIS.BOX Button 1 is red;
- Steps 6–7: the duration $b(k)^m$ in which the operational LED color of the i -th forklift KIS.BOX Button 2 is red.

As in the previous section, the data concerning the performance of the above system can be gathered using the Data trend chart. These data include a timestamp which clearly identifies the working time within the shift. Subsequently, the obtained data can be merged with the detailed description of all items (6.13) as well as operator experience, which is associated with the KIS.BOX being used. As a result, $k = 0, \dots, n_E$ Datapoints are obtained, which can be used for designing a model of the forklift operator's performance:

$$L_f(k) = \{t(k), s(k), d(k), m(k), q(k), f(k)^m, b(k)^m, e_x(k), t_s(k)\}, \quad (6.14)$$

where $e_x(k)$ and $t_s(k)$ are respectively experience and working time within the shift of the forklift operator transporting the k -th item.

Similarly as in the preceding section, two TS models will be developed. The fuzzy model is divided into two sub-models:

$$f = f_f(d, m, q, e_x, t_s), \quad (6.15)$$

$$b = f_b(d, m, q, e_x, t_s). \quad (6.16)$$

Finally, $f_f(\cdot)$ and $f_b(\cdot)$ signify a given model structure used for calculating either $f(k)$ or $b(k)$ for the k -th item. Owing to safety purposes, the maximum velocity limit can be imposed, e.g., at the level of $v_f = 5 \text{ km/h} = 1.39 \text{ m/s}$:

$$f \geq \frac{1}{v_f}d, \quad (6.17)$$

which is implied by the fact that $distance = time \times velocity$. The same constraint is imposed on b , i.e., $f \geq \frac{1}{v_f}d$. It is also evident that the mass m and volume q of an item have a direct impact on f .

Similarly as in the preceding section, two fuzzy premise variables are introduced, which are detailed in Table 6.3. Under these preliminaries, the TS model of (6.15) is

$$\begin{aligned} &\text{IF } t_s \in \mathbb{T}_{s,j} \text{ and } e_x \in \mathbb{E}_{x,i}, \text{ THEN} \\ &f = p_{1,f}^{i,j}d + p_{2,f}^{i,j}m + p_{3,f}^{i,j}q, \quad i, j = 1, \dots, 3, \end{aligned} \quad (6.18)$$

while (6.16) obeys

$$\begin{aligned} &\text{IF } t_s \in \mathbb{T}_{s,j} \text{ and } e_x \in \mathbb{E}_{x,i}, \text{ THEN} \\ &b = p_{1,b}^{i,j}d + p_{2,b}^{i,j}m + p_{3,b}^{i,j}q, \quad i, j = 1, \dots, 3, \end{aligned} \quad (6.19)$$

where p_i are the model parameters which have to be determined while $\mathbb{E}_{x,j}$ and $\mathbb{T}_{s,i}$ are fuzzy sets associated with Table 6.3. Using the rule firing strength (6.6), the models (6.18)–(6.19) can be written as follows:

$$f = \sum_{i=1}^3 \sum_{j=1}^3 \mu_{i,j} \left(p_{1,f}^{i,j}d + p_{2,f}^{i,j}m + p_{3,f}^{i,j}q \right), \quad (6.20)$$

$$b = \sum_{i=1}^3 \sum_{j=1}^3 \mu_{i,j} \left(p_{1,b}^{i,j}d + p_{2,b}^{i,j}m + p_{3,b}^{i,j}q \right). \quad (6.21)$$

The final set boils down to transforming (6.20)–(6.21) to the regressor form:

$$f = r_f^T p_f, \quad (6.22)$$

where

- $r_f = [\mu_{1,1}r_f^T, \mu_{1,2}r_f^T, \dots, \mu_{2,2}r_f^T]^T$,
- $p_f = [(p_f^{1,1})^T, (p_f^{1,2})^T, \dots, (p_m^{3,3})^T]^T$,

and $r_f = [d, m, q]^T$. The model (6.21) can be formulated in a similar way:

$$b = r_b^T p_b, \quad (6.23)$$

where

- $r_b = [\mu_{1,1}r_b^T, \mu_{1,2}r_b^T, \dots, \mu_{2,2}r_b^T]^T$,
- $p_b = [(p_b^{1,1})^T, (p_b^{1,2})^T, \dots, (p_b^{3,3})^T]^T$.

The objective of the remaining part of this section is to provide a concise outline of the general algorithm for designing (6.22)–(6.23):

KIS.BOX assignment: Assign forklifts operators with different experience (see Table 6.3) to different KIS.BOXes.

Transportation plan: Obtain the transportation plan (6.13) consisting of n_E transportation events.

Data gathering: Perform the data gathering procedure according to the above 8-step strategy:

- obtain the data from the Data trend chart;
- synchronize the obtained data with the transportation plan;
- based on the KIS.BOX assignment, extend the data set with the forklift operator's experience.

Model design: Use the recursive least square algorithm to determine (6.22)–(6.23).

> Highlights

The model (6.22)–(6.23) may have several crucial applications. Having a transportation plan (6.13), one can use (6.22)–(6.23) to

1. schedule transportation of items to the transfer stations and estimate its feasibility;
2. schedule the work of individual forklift operators, which will minimize a possible bottle neck effect at the transfer stations;
3. make the transportation process transparent, and hence increase various KPIs like the ones proposed in Sect. 5.1;
4. use it as a digital twin of the forklift operators, which can be employed for their training and permanent skill improvement.

6.3 Integrating Workers Within a Semi-automatic Assembly System

A sample general semi-automatic assembly scheme is presented in Fig. 6.2. Indeed, it may be surrounded by conveyor belts or other transportation means that provide suitable components for the assembly process. Let us begin with recalling the fact that the worker can start mounting the next battery iff the previous one is completed. Let us also recall (see Sect. 6.1.1) that the completion time is the sum of

1. battery cell mounting time c_m ;
2. cell-controller linking time c_l .

Let us define the following variables (cf. Table 6.2 and Fig. 6.2):

- $x(k)$: start time of mounting the k -th battery,
- $u_m(k)$: cell delivery time for the k -th battery,
- $u_c(k)$: controller delivery time for the k -th battery.

This nomenclature allows formulating a simple formal description pertaining to the evolution of the start time of mounting consecutive batteries:

$$x(k + 1) = \max(x(k) + c_m(k) + c_l(k), u_m(k + 1), u_c(k + 1)), \tag{6.24}$$

where $c_m(k), c_l(k)$ are respectively c_m and c_l for the k -th battery. This means that the maximum of the times included in (6.24), i.e., in $\max(\cdot)$, determines the start time of mounting the $k + 1$ st battery.

Checking manufacturing feasibility

Without any loss of generality, let us assume that the starting assembly time is $x(0) = 0$, which may correspond to 8.00 h. Moreover, let the battery cells and the controller be available at that time, i.e., $u_c(0) = 0$ and $u_l(0) = 0$.

There is a set of 10 different batteries to be mounted. Having the worker experience, as well as required parameters of the batteries described along with (6.1)–(6.2) and the performance model (6.10)–(6.11), one can calculate the expected battery mounting times $c_m(k) + c_l(k)$, which are given in Table 6.5. The cells and the controller are delivered to the assembly station every 19 and 20 min, respectively. The problem is to check if there is a chance to mount all 10 batteries in 200 min. For that purpose, one can use (6.24). As an example let us employ it for calculating $x(1)$, i.e., the start time of mounting battery number 1 (Table 6.6):

$$x(1) = \max(x(0) + c_m(0) + c_l(0), u_c(1), u_l(1)) = \max(23, 19, 20) = 23. \tag{6.25}$$

As a result, the mounting start time for the last battery is $x(9) = 184$ while its expected assembly time is 20 min, and hence all batteries will be completed in 204 min.

Table 6.5 Expected battery mounting times

k -th Battery	$c_m(k) + c_l(k)$
0	23
1	19
3	16
4	20
5	15
6	27
7	12
8	20
9	24

Table 6.6 Expected battery mounting times

k -th Battery	$c_m(k) + c_l(k)$	$x(k)$	$u_c(k)$	$u_l(k)$
0	23	0	0	0
1	19	23	19	20
2	16	42	38	40
3	20	60	57	60
4	15	80	76	80
5	27	100	95	100
6	12	127	114	120
7	20	140	133	140
8	24	160	152	160
9	20	184	171	180

➤ Towards flexible manufacturing

A direct calculation of the total battery assembly time,

$$T_{\text{total}} = \sum_{k=0}^9 (c_m(k) + c_l(k)) = 196 \text{ [min]}, \quad (6.26)$$

clearly indicates that it is possible to perform the above task in 200 min. This, however, requires a different strategy for calculating component delivery times $u_m(k)$ and $u_c(k)$. Indeed, there are different types of batteries with different expected assembly times (cf. Table 6.5). Thus, because the employed delivery rate is constant, it prevents attaining optimal assembly performance.

To settle this important issue, it is proposed to extend the scheme presented in Fig. 6.2 with an external device and scheduling software. As a result, the scheme shown in Fig. 6.5 is obtained. The purpose of the external software is to calculate optimal component delivery times $u_m(k)$ and $u_c(k)$, which will make it possible to have the components just on time. The software should also take into account inevitable delays which can be caused by the worker. Additionally, such delays should be compensated for as much as possible. For further details the reader is referred to the authors' publications concerning such developments [11, 16, 17]. Indeed, using the fault-tolerance concept, these approaches can also minimize various unappealing phenomena like, e.g., delays. Additionally, the control process involves a cost function, which can take into account a wide range of economic aspects [14].

As can be seen in Fig. 6.5, the hardware is communicated with KIS.BOX through its digital outputs, which provide information about the current assembly status. Finally, the scheduling sequence is employed to control the item delivery process.

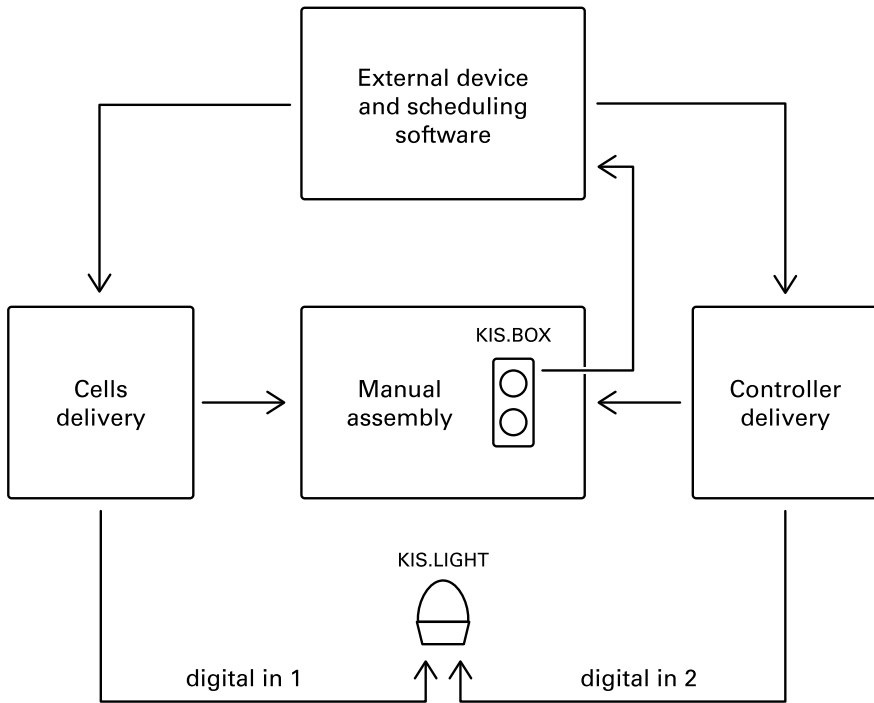


Fig. 6.5 General overview of the manual assembly station with the KIS.ME infrastructure as well as external hardware and software

6.4 Scheduling Transportation Actions

In Chap. 3 and Sect. 5.1, a set of logistic and transportation solutions were proposed along with measures that can be used for assessing their performance. However, in all cases the decisions concerning the required transportation actions were made either by the transporter operator or the KIS.MANAGER operator. In this section, the warehouse transportation system proposed in Sect. 6.2 is extended with a scheduling framework, which makes it possible to deliver the items (cf. Fig. 6.3) just on time. Similarly as in Sect. 6.3, let us introduce the following variables concerning the i -th forklift:

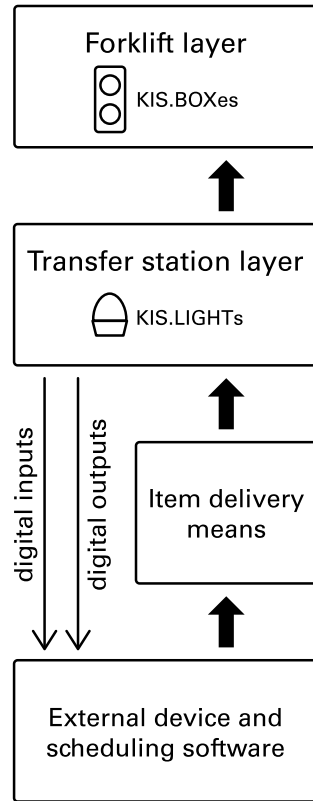
- $x_i(k)$: the start time of collecting the k -th item from the $T(k)$ transfer station,
- $u(k)$: the delivery time of the k -th item to the $T(k)$ transfer station,
- $v_i(k)$: a decision variable associating the k -th item with the i -th forklift.

The decision variable may have two states:

$v_i(k) = 1$: the k -th item is transported by the i -th forklift,

$v_i(k) = 0$: the k -th item is not transported by the i -th forklift.

Fig. 6.6 General overview of the transportation system with the KIS.ME infrastructure as well as the external hardware and software



A scheme of the proposed general transportation framework is presented in Fig. 6.6. From the KIS.MANAGER viewpoint, the proposed scheme extends the one discussed in Sect. 6.2 by introducing rules of the following form: If a forklift performed transportation of the k -th item, then the false–true–false sequence is sent to KIS.LIGHT digital output 1 associated with the $T(k)$ transfer station. This slight modification allows gathering measurements pertaining to the transportation time of the k -th item. Indeed, this process starts from the “None” state (see Table 6.4) of the KIS.LIGHT associated with the $T(k)$ transfer station and ends with the false–true–false sequence on its digital output 1.

Under the above preliminaries, the time-driven model of the i -th forklift is given by

$$x_i(k + 1) = \max (x_i(k) + f_{d,i}(x) + b_{d,i}(k), u(k + 1)v_i(k + 1)), \quad (6.27)$$

where

$$f_{d,i}(k) = f_i(k)v_i(k), \quad (6.28)$$

$$b_{d,i}(k) = b_i(k)v_i(k), \quad (6.29)$$

while $f(k)$ and $b(k)$ can be calculated using the model (6.22)–(6.23) developed in Sect. 6.2. This means that, depending on the decision $v_i(k)$, the variables $f_{d,i}(k)$ and $b_{d,i}(k)$ can be equal to either 0 or $f_i(k)$ and $b_i(k)$, respectively.

Two forklifts example

Let us consider two forklifts and one transfer station located in front of the aisle corresponding to the second forklift. Figure 6.3 shows the second forklift and the transfer station with the KIS.LIGHT operational LED lighting in magenta. The first forklift operates in front of the transfer station with the KIS.LIGHT operational LED lighting in yellow. Let us imagine that the item is delivered to the transfer station (magenta) and it has to be transported by the first forklift. This means that the corresponding KIS.LIGHT digital input is set to the “Left hand side” state (cf. Table 6.4). Based on such information, the external software should set the decision variables in the following way: $v_1(0) = 1$ and $v_2(0) = 0$, which clearly indicates that the first forklift has to perform transportation of the 0th item. Let us also assume that $x_1(0) = 0$ and $x_2(0) = 0$, which may correspond to 8.00, i.e., the beginning of the shift. Moreover, let $f_1(0) = 3$ and $b_1(0) = 2$ minutes. Thus, Eqs. (6.27)–(6.29) imply that

$$x_1(1) = \max(0 + 3 + 2, u(1)v_1(k)) = \max(5, u(1)v_1(1)), \quad (6.30)$$

$$x_2(1) = \max(0, u(1)v_2(1)) = u(1)v_2(1). \quad (6.31)$$

From the above results, it is evident that, if the subsequent, i.e., $k = 1$, item should be transported by the first forklift ($v_1(1) = 1$ and $v_2(1) = 0$), then $u(1) \leq 5$ guarantees just-on-time performance. Contrarily, if the subsequent, i.e., $k = 1$, item should be transported by the first forklift ($v_1(1) = 0$ and $v_2(1) = 1$), then $u(1) = 0$ guarantees just-on-time performance.

The above preliminary results formed the basis for formulating the scheduling strategies proposed by the authors [1, 11, 14, 15, 17], which can be efficiently used for various kinds of problems. The crucial element of these strategies is the cost function:

$$J = \sum_{k=0}^{n_E-1} u(k), \quad (6.32)$$

which has to be maximized for all n_E transportation events (6.13) taking into account (6.27) and the related constraints [1, 11, 14, 15, 17]. The process of maximizing (6.32) can be interpreted as finding the just-on-time item delivery times $u(k)$ for the forklift transportation system.

6.4.1 Health-aware and Fault-Tolerant Transportation Scheduling

The scheduling software presented in Fig. 6.6 can be extended with additional elements, which can settle the following problems:

- The proposed transportation system is a time-driven one, i.e., all modelled and optimized variables represent certain operation times. From the classical system control viewpoint [12, 18], the fault is defined as an unpermitted deviation of at least one characterised property of the system compared to its nominal value. Thus, delays within transportation system can be interpreted as faults [11, 14–17]. This justifies the need for implementing fault-tolerance in the transportation system.
- The application of indoor-operating forklifts makes it necessary to use various kinds of batteries or accumulators. Thus, while scheduling the work of forklifts one has to take into account a compulsory recharging process as well as the state of charge (SOC) and the state of health (SOH) of the batteries.

Before settling the above issues, it is necessary to provide a list of crucial constraints which influence the performance of the transportation system [14]:

Transfer constraint: For the same transfer station, it is possible to deliver a new item iff the previous one has been removed. Thus, for all l and k ,

$$\text{IF } l > k \text{ and } T(k) = T(l) \text{ THEN } u(k) \leq u(l) + t(j), j < n_E, \quad (6.33)$$

where $t(j) > 0$ is a unknown time between delivering the k -th and l -th items to the $T(k) = T(l)$ transfer station.

Scheduling constraint: This constraint simply states that all n_E starting times of transportation events should be bounded:

$$u(k) \leq u_E, \quad (6.34)$$

where $u_E > 0$ is maximum scheduled time for starting all n_E transportation events.

Energy constraint: It is assumed that each forklift is equipped with the battery management system (BMS), which provides information about the maximum operational time of the i -th forklift $\tau_i(k)$ for the k -th transportation system.

Let us start with analysing potential faulty situations, which will have direct influence on the feasibility of constraints. The delay in taking the k -th item will cause $t(j)$ to be extended, i.e., the delivery of the subsequent l -th item will be delayed. Thus, with

appropriately large $t(j) > 0$, the constraint (6.33) is always feasible. Contrarily, a faulty situation may cause (6.34) to be infeasible. Thus, fault-tolerance can be attained by introducing a relaxation variable $\bar{u} \geq 0$, which should be as small as possible. Additionally, if the faulty situation makes the measured transportation times larger than the nominal values, then they should replace them in the model (6.27) of the i -th forklift, which suffers from the delay. Thus, to achieve fault-tolerance, the minimized cost function should be given as follows:

$$J = -p_1 \sum_{k=0}^{n_E-1} u(k) + p_2 \bar{u}, \quad (6.35)$$

where $p_1 \geq 0$ and $p_2 \geq 0$, $p_1 + p_2 = 1$ are weighting parameters, which can be used for attaining a balance between fault-tolerance and performance.

Let us proceed to health-aware analysis. For simplicity, it is assumed that each forklift is operating in a single aisle. However, more flexibility can be attained by removing this constraint. Thus, one can imagine that the tasks with shorter (smaller $f_i(k) + b_i(k)$) and less energy-demanding transportation tasks are realized by forklifts with smaller $\tau_i(k)$. This can be formally expressed by the following cost function:

$$J_H = - \sum_{k=0}^{n_E-1} \sum_{i=1}^{n_F} (\tau_i(k) - b_i(k) - c_i(k)), \quad (6.36)$$

where n_F is the number of forklifts. Finally, the minimized cost function takes the form

$$J = -p_1 \sum_{k=0}^{n_E-1} u(k) + p_2 \bar{u} + p_3 J_H, \quad (6.37)$$

where $p_3 \geq 0$, $p_1 + p_2 + p_3 = 1$.

➤ Economy oriented cost function

The cost function (6.37) is very general and can be extended with additional components corresponding to the operational cost of the transportation system. This is especially important in the case of a fleet consisting of different forklifts. Thus, each forklift may involve different operational costs.

6.5 Training Exercise: Work Scheduling

6.1 Scheduling tasks of three workers

Exercise requirements: The exercise requires access to one KIS.LIGHT, three KIS.BOXes and a photoelectric sensor with an M12 8 pin interface (see Sect. 3.1).

1. There are three workers that can perform the same kind of tasks using the component sets provided by the conveyor belt (see Fig. 6.7).
2. The conveyor belt transfers a component set to the beginning of the working area and then stops.
3. Use the photoelectric sensor to detect the arrival of the component set and feed this information to KIS.LIGHT digital input 1.
4. Each worker operates according to the states presented in Table 6.7.
5. Each worker has its own individual KIS.BOX. Implement the state-space model (6.7) for each KIS.BOX (see Sect. 2.10). As a trigger use an action associated with pressing KIS.BOX Button.
6. Each worker can be either present or absent at the workplace. Implement the state-space model (Table 6.8) for each KIS.BOX. As a trigger use an action associated with pressing KIS.BOX Button 2.
7. Depending on the KIS.LIGH digital outputs' state (see Table 6.9), the conveyor belt can move a component set to one of the three workplaces.
8. using Rule engine Implement the system logic in such a way that new component sets are delivered to the workers being present and in an idle state.

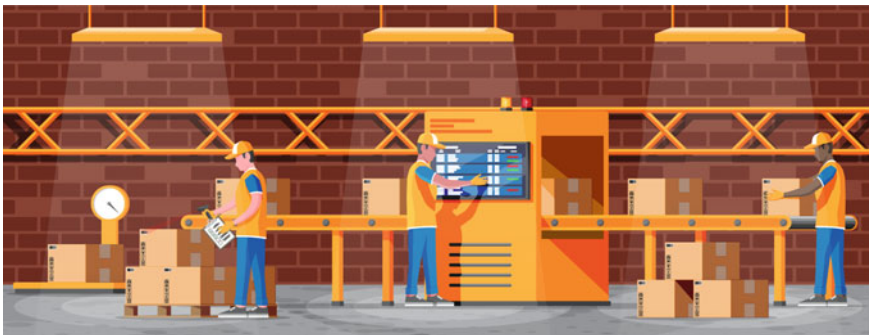


Fig. 6.7 Scheduling tasks of three workers

Table 6.7 Worker's KIS.BOX Button 1 operational LED states

State	KIS.BOX Button 1 operational LED color
Idle	Green
Mounting	Red
Packing	Yellow

Table 6.8 Worker's KIS.BOX Button 2 operational LED states

State	KIS.BOX Button 1 operational LED color
Present	Green
Absent	Red

Table 6.9 KIS.LIGHT transportation actions

Worker	KL digital output 1	KL digital output 2
1	0	1
2	1	0
3	1	1

6.6 Concluding Remarks

The main objective of this chapter was to provide a selected list of motivating examples pertaining to potential applications of KIS.ME. Although the presented instances concern a dedicated application study, they can be relatively easily adapted to different applications frequently encountered in various industries. In particular, it was shown how to model human operator performance both in assembly and transportation tasks. The resulting model is particularly important for various kinds of planning and scheduling. Indeed, such models make it possible to obtain transparency and predictability of the controlled and monitored system. Additionally, as most indoor transportation means employ various kinds of accumulators, it is profitable to take into account their health. Thus, it was shown how to include such information in the scheduling procedure. As a result, the overall accumulator health of the transporter fleet can be balanced. Another important aspect is that the human operated system may suffer from various kinds of delays. Thus, it was shown how to attain fault-tolerance which can prevent such a kind of unappealing situations. Finally, the chapter ended with advanced training exercises, which pertain to work scheduling of three workers operating a conveyor belt.

References

1. M. Witczak, B. Lipiec, M. Mrugalski, L. Seybold, Z. Banaszak. Fuzzy modelling and robust fault-tolerant scheduling of cooperating forklifts, in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE, Glasgow, 2020), pp. 1–10
2. L.A. Zadeh, Fuzzy logic. *Computer* **21**(4), 83–93 (1988)
3. G. Klir, B. Yuan, *Fuzzy Sets and Fuzzy Logic* (Prentice Hall, New Jersey, 1995), vol. 4
4. P. Hájek, *Metamathematics of Fuzzy Logic* (Springer Science & Business Media, Berlin, 2013)
5. Z. Nivolianitou, M. Konstantinidou, *A fuzzy modeling application for human reliability analysis in the process industry, in Human Factors and Reliability Engineering for Safety and Security in Critical Infrastructures* (Springer, Berlin, 2018), pp.109–154

6. M. Konstandinidou, Z. Nivolianitou, Ch. Kiranoudis, N. Markatos, A fuzzy modeling application of CREAM methodology for human reliability analysis. *Reliab. Eng. Syst. Saf.* **91**(6), 706–716 (2006)
7. Q. Zhou, Y.D. Wong, H.S. Loh, K.F. Yuen, A fuzzy and bayesian network CREAM model for human reliability analysis—the case of tanker shipping. *Saf. Sci.* **105**(6), 149–157 (2018)
8. R. Sujatha, V. Sharmila, P. Hema, M. Amruth Varshinee, A. Prasnath. Some aspects of human error analysis using fuzzy relations. *Int. J. Pure Appl. Math.* **117**(21), 311–325 (2017)
9. M.A. Segura, S. Hennequin, B. Finel, Human factor modelled by fuzzy logic in preventive maintenance actions. *Int. J. Operat. Res.* **27**(1–2), 316–340 (2016)
10. G. Suprakash, K. Pramod, Y.R. Gunda, A fuzzy causal relational mapping and rough set-based model for context-specific human error rate estimation. *Int. J. Occup. Saf. Ergon.* **27**(1), 63–78 (2021)
11. L. Seybold, M. Witczak, P. Majdzik, R. Stetter, Towards robust predictive fault-tolerant control for a battery assembly system. *Int. J. Appl. Math. Comput. Sci.* **25**(4), 849–862 (2015)
12. M. Witczak, *Fault Diagnosis and Fault-Tolerant Control Strategies for Non-Linear Systems: Analytical and Soft Computing approaches* (Springer International Publishing, Heidelberg, Germany, 2014)
13. O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks, Fuzzy Models, and Gaussian Processes* (Springer Nature, Berlin, 2020)
14. M. Witczak, Lo. Seybold, G. Bocewicz, M. Mrugalski, A. Gola, Z. Banaszak. A fuzzy logic approach to remaining useful life control and scheduling of cooperating forklifts, in *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE, Luxemburg, 2021), pp. 1–8
15. M. Witczak, P. Majdzik, R. Stetter, B. Lipiec, A fault-tolerant control strategy for multiple automated guided vehicles. *J. Manuf. Syst.* **55**(4), 56–68 (2020)
16. P. Majdzik, M. Witczak, B. Lipiec, Z. Banaszak. Integrated fault-tolerant control of assembly and automated guided vehicle-based transportation layers. *Int. J. Comput. Integr. Manuf.* 1–18 (2021)
17. P. Majdzik, A. Akielaszek-Witczak, L. Seybold, R. Stetter, B. Mrugalska, A fault-tolerant approach to the control of a battery assembly system. *Control Eng. Pract.* **55**(10), 139–148 (2016)
18. M. Witczak, *Modelling and Estimation Strategies for Fault Diagnosis of Non-linear Systems* (Springer, Berlin, 2007)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Chapter 7

KIS.API: Towards External Communication



7.1 Introduction to KIS.API

KIS.API is defined as a particular instance of REST API [1–3], which is designed for the purpose of unified and convenient communication with KIS.Devices and the associated environment operating within KIS.MANAGER.

The crucial components of KIS.API are the resources, which can be defined in a very broad sense. In fact, anything having a name can be perceived as a resource, e.g., a user, a workspace, an asset, etc. The crucial information associated with a resource pertains to the service being realized, i.e., the transfer of data as well as the associated actions. A general structure of the resource is given in Table 7.1 [1].

Let us start with the . As an example, one can consider a KIS.Device, which can be characterized by, e.g.,

- an ID,
- a name,
- a URN,
- associated asset group (workspace) IDs,

while its possible representation can be given in an intuitive JSON [1] form:

```
{
  "id": 10102,
  "urn": "urn:rafi:sbox:9c65f93cbcd6",
  "name": "KIS.BOX_9C65F93CBED6",
  "assetGroupIDs": [
    9757,
    9758
  ]
}
```

Table 7.1 A typical structure of a resource

Property	Description
Representation	The way and structure of the data representation, e.g., JSON, XML
Identifier	A URL that refers to a specific resource at any time
Metadata	The content type, modification time, etc
Control data	Any data identifying the status of a resource, e.g., last modification date

Table 7.2 KIS.API actions

HTTP verb	Action
GET	Access a resource in a read-only way
POST	Send a new resource
PUT	Update a resource
DELETE	Delete a resource

Table 7.3 KIS.API response

Status code	Description
200	Successful response
204	No content returned
400	Bad request structure
404	Unsuccessful response

Now let us proceed to the resource identifier, which provides a unique way to identify the resource at any time instance. In other words, it should provide a full and unique path to the resource. Since the REST structure is based on HTTP, a sample path can look as follows:

```
https://api.kisme.com/kisapi/v1/assets/assetUrn,
```

which uniquely identifies a given KIS.Device using its URN. Having a resource identifier, one can proceed to realize some actions with it. A general set of verbs, which defines specific actions, is given in Table 7.2. Finally, an important standard that is inherited by REST from HTTP pertains to the status code. The most common status codes are given in Table 7.3. Under the above preliminary information, one can proceed to the registration and authorization of a new KIS.API user using KIS.MANGER.



7.1.1 User Registration and Authorization

The primary objective of this part is to define a new KIS.API user along with an appropriate credentials. The process starts with selecting the Main menu → Portal

KIS.API Credentials



Fig. 7.1 Defining KIS.API credentials

admin and then pressing the KIS.API icon . Subsequently, a new KIS.API user can be created with . The process is fully automatic and the only information required is to provide a user description, i.e., a name. A sample KIS.API credentials generation process is presented in Fig. 7.1. As a sample, a KIS.API access control triplex is obtained, which can be summarized as follows:

- the client ID,
- the API key,
- the baseUrl.

Note that baseUrl is simply the base resource identifier detailed in the preceding section, i.e.,

```
https://api.kisme.com/kisapi/v1/
```


The objective of the subsequent sections is to provide a concise introduction to KIS.API. Thus, to simplify this process, the Postman (<https://www.postman.com/>) application is employed. It is a dedicated platform for building and using APIs. In other words, Postman can be perceived as an HTTP client for testing web services, which makes it easy to test APIs by providing a simple interface for issuing API requests and viewing responses. The Postman registration and configuration process is very intuitive, and hence it is omitted. The subsequent step is to proceed to the KIS.API documentation (<https://docs.kisme.com>) and then select Run are Postman button. As a result, KIS.API collection is loaded into the Postman workspace. The final step is to provide the above defined KIS.API credentials using . This process is illustrated in Fig. 7.2. As of that moment, all KIS.API commands can be accessed and tested using the intuitive Postman graphical user interface. This process boils down to selecting an appropriate option and then sending a desired request to KIS.API. Thus, the objective is to provide a concise review of all available KIS.API functionalities.


Table 7.4 List of possible requests associated with an asset

No.	Verb	Path	Action
1	GET	baseUrl/assets	Obtain a list of all assets
2	GET	baseUrl/assets/assetUrn	Obtain the asset device information
3	PUT	baseUrl/assets/assetUrn	Update the name of the asset
4	PUT	baseUrl/assets/assetUrn/assetgroupId/ assetgroups	Add the asset to an asset group
5	DEL	baseUrl/assets/assetUrn/assetgroupId/ assetgroups	Remove the asset from an asset group

Table 7.5 List of possible requests associated with an asset group

No.	Verb	Path	Action
1	GET	baseUrl/assetgroups	Obtain a list of all asset groups
2	GET	baseUrl/assetgroups/assetgroupId	Obtain the asset group information
3	PUT	baseUrl/assetgroups/assetgroupId	Update the asset group information

assetGroupIDs: an array containing numerical values of the asset groups associated with the asset.

Note that most of the above features can be directly accessed through KIS.MANAGER by selecting Main menu → Assets and then choosing a desired asset. Subsequently, the information about the asset can be retrieved by pressing  (see Fig. 2.16).

Having all the above information, let us proceed to two simple examples of using the discussed requests.

Obtaining a list of assets

This example concerns a response to the request no. 1 listed in Table 7.4. As a result, the following JSON structure can be obtained:

```

]
  {
    "id": 10102,
    "urn": "urn:rafi:sbox:9c65f93cbcd6",
    "name": "KIS.BOX 9C65F93CBED6",
    "assetGroupIDs": [
      9757,
      9758
    ]
  },
  {
    "id": 10153,

```



```

    "urn": "urn:rafi:sbox:9c65f93cbeb8",
    "name": "KIS.BOX 9C65F93CBEB8",
    "assetGroupIDs": [
        9757,
        9758
    ]
}
]

```

As can be observed, the response contains information about two KIS.BOXes, which are assigned to two asset groups (9757 and 9758).

Removing an asset from the asset group

This example concerns a response to the request no. 5 listed in Table 7.4. Let us consider the first asset (KIS.BOX) given in the preceding example. Its URN is `urn:rafi:sbox:9c65f93cbcd6`, and it is assigned to two asset groups 9757 and 9758. The objective is to remove it from the asset group 9757, and hence the `assetgroupId` parameter should be set to 9757. As a result, the following JSON structure can be obtained:

```

{
  "message": "The operation is in conflict with the
relationship constraint 'assetsMustBeMemberOfInventory'",
  "code": 409
}

```

which simply means that it is impossible to remove an asset from the list of all available assets (see Table 2.5 for a comprehensive explanation). Thus, let us change the `assetgroupId` parameter to 9758 (the second available asset group). As a result, the following JSON structure is obtained:

```

{
  "id": 10102,
  "urn": "urn:rafi:sbox:9c65f93cbcd6",
  "name": "KIS.BOX 9C65F93CBED6",
  "assetGroupIDs": [
    9757
  ],
  "isOnline": false,
  "hardware": {
  },
  "software": {

```

```

    },
    "certificate": {
    },
    "network": {
    },
    "firmwareUpdate": {
    }
  }
}

```

As can be observed, the assignment of this asset to the asset group 9758 was removed.

Let us proceed to the asset groups for which the available request list is given in Table 7.5. As can be observed, most requests require an additional parameter `assetgroupId`, which can be retrieved through the request no. 1 in Table 7.5. The response pertaining to the request detailed in Table 7.5 may contain the following parameters:

- ID: the asset group identifier,
- `assetIDs`: an array containing numerical values of asset IDs associated with the asset group,
- `name`: the name of the asset,
- `isOnline`: the asset `isOnline` Datapoint value,
- `description`: a detailed description of the asset group,
- `name`: the name of the asset group.

Let us proceed to two simple examples explaining the application of the above requests.

Obtaining a list of asset groups

This example concerns a response to the request no. 1 listed in Table 7.5. As a result, the following JSON structure can be obtained:

```

[
  {
    "id": 9757,
    "name": "My Devices",
    "assetIds": [
      10102,
      10153
    ]
  },
  {
    "id": 9758,

```

```

        "name": "Workspace 1",
        "assetIds": [
            10153
        ]
    }
]

```

As can be observed, the response contains information about two asset groups and the assets associated with them (IDs: 10102, 10153).

Updating the asset group description

This example concerns a response to the request no. 3 listed in Table 7.5. Let us provide a new description of Workspace 1 (ID 9758) in the JSON form:

```

{
    "name": "Workspace 1",
    "description": "Main Workspace"
}

```

In the case of the Postman application, to provide such a description one should go to the `Body` tab of the request and enter the above JSON structure. As a result, the following JSON structure can be obtained:

```

{
    "id": 9758,
    "name": "Workspace 1",
    "assetIds": [
        10153
    ],
    "description": "Main Workspace"
}

```

Having access to assets and the associated asset groups, let us proceed to the user management functionalities, which are listed in Table 7.6. As can be observed in Table 7.6, requests no. 2 and 3 require an additional path parameter called `accountNumber`. Note that in KIS.MANAGER the user is identified by its name and email. Thus, at least one of these parameters should be known while realizing the request no. 1 in Table 7.6. Thus, the obtained response can be used to obtain the associated `accountNumber`.

Table 7.6 List of possible requests associated with users

No.	Verb	Path	Action
1	GET	baseUrl/users	Obtain a list of all users
2	GET	baseUrl/users/accountNumber	Obtain user information
3	DEL	baseUrl/users/accountNumber	Delete a user

Displaying all users and their accountNumber

This example pertains to realisation of the request no. 1 in Table 7.6. As a result, the following JSON structure can be obtained:

```
[
  {
    "name": "Jack Cactus",
    "email": "j.cactus@controlintech.pl",
    "accountNumber": "3d5997bb-f6ee-4681-8adf-0ce7366e2964"
  },
  {
    "name": "Hans Wurst",
    "email": "h.wurst@controlintech.pl",
    "accountNumber": "4f4009c2-1a7b-4531-b780-702a19cd62a1"
  }
]
```

The structure contains information about two users and their associated accountNumber.

Deleting a user

The JSON structure obtained in the preceding example contains information about two users. The objective of the current example is to delete the user identified:

```
"accountNumber": "3d5997bb-f6ee-4681-8adf-0ce7366e2964"
```

For that purpose, the above number has to be provided as a path parameter in the Postman application. Note that, after sending a request to KIS.API, no JSON structure is received (cf. code 204 in Table 7.3).

Table 7.7 List of possible requests associated with Datapoints

No.	Verb	Path	Action
1	GET	BaseUrl/assets/assetUrn/ datapointDefinitions	Obtain a list of Datapoints
2	GET	BaseUrl/assets/assetUrn/ datapointDefinitions/datapointValues	Obtain Datapoint values

7.2.2 Accessing Data Through Datapoints

The objective of this point is to provide a way of accessing the data associated with Datapoints. For a comprehensive description of Datapoints, the reader is referred to Sect. 2.7 and Appendix. B. The possible requests associated with Datapoints are provided in Table 7.7. It should be also noted that the above requests require the following path parameters:

- `assetUrn`: directly printed on an asset (KIS.Device), can be retrieved through the request no. 1 from Table 7.4 or through KIS.MANAGER;
- `datapointDefinition`: the Datapoint name, e.g., `button1Pressed`.

Additionally, request no. 2 in Table 7.7 can be executed with the following query parameters:

- `from`: an ISO timestamp indicating a lower bound of the required time range;
- `to`: an ISO timestamp indicating an upper bound of the required time range;
- `limit`: a maximum number of required Datapoint values.

Note that it is not compulsory to use all of the above-listed query parameters simultaneously. For example, the `limit` parameter can be employed as a standalone one.

Obtaining a list of Datapoints

The example is concerned with the request no. 1 in Table 7.7. As a result of using it, a full list of Datapoints along with their types is returned. A sample form of such a couple is given as follows:

```
{
  "name": "button2Pressed",
  "datatype": "BOOLEAN"
}
```

Obtaining five recent Datapoint values

For the purpose of this example, a set of two rules is implemented (see Sect. 2.9 for more details), i.e.,

1. If the KB Button1 operational LED color is black, then the KB Button 1 operational LED color is red.
2. If the KB Button 1 operational LED color is red, then the KB operational LED color is black.

Thus, the purpose of the above rules is to switch the KB Button1 operational LED color from red to black (no illumination) and vice versa. This means that the resulting effect should be the KB Button 1 operational LED blinking in red. However, to achieve such an effect, the KB Button 1 operational LED color should be initiated using its digital twin (see Sect. 2.6) by setting the above color to either black or red. Subsequently, the path parameters should be defined, i.e., `assetUrn` and `datapointDefinition`. The latter one is set to `button1ColorKpi`. Finally, the query parameter `limit` is set to five. As a result, a JSON structure is obtained containing the five recent values of the indicated Datapoint:

```
[
  {
    "timestamp": "2022-08-30T11:50:07.478Z",
    "value": "2"
  },
  {
    "timestamp": "2022-08-30T11:50:06.478Z",
    "value": "5"
  },
  {
    "timestamp": "2022-08-30T11:50:05.524Z",
    "value": "2"
  },
  {
    "timestamp": "2022-08-30T11:50:04.712Z",
    "value": "5"
  },
  {
    "timestamp": "2022-08-30T11:50:03.649Z",
    "value": "2"
  }
]
```

The recorded Datapoint values simply indicate that the KB Button 1 operational LED changes its color from red (5) to black (2) and vice versa. As can be observed,

the switching process takes more or less one second. However, this time is data transfer-dependent, and hence its is not uniform.

Obtaining five recent Datapoint values from a given time frame

The objective of this example is to focus on the reader attention to ISO data-time format, which is given, e.g., by

```
2022-08-30T14:02:07.478Z
```

Its construction is obvious: however, it contains characters which are not permitted in a URL construction, i.e. '.', which should be simply replaced by its equivalent equal to '%3A', yielding

```
2022-08-30T14%3A02%3A07.478Z
```

Thus, by setting the `from` query parameter according to the above form one can obtain:

```
[
  {
    "timestamp": "2022-08-30T14:02:32.304Z",
    "value": "5"
  },
  {
    "timestamp": "2022-08-30T14:02:31.304Z",
    "value": "2"
  },
  {
    "timestamp": "2022-08-30T14:02:30.336Z",
    "value": "5"
  },
  {
    "timestamp": "2022-08-30T14:02:29.507Z",
    "value": "2"
  },
  {
    "timestamp": "2022-08-30T14:02:28.475Z",
    "value": "5"
  }
]
```

7.2.3 KPIs and Calculated Datapoints

This section constitutes a continuation of the preceding one. Indeed, CDPs and KPIs (see Sect. 4.1.2 and Appendices A and B) employ Datapoints as a basis for forming desired answers pertaining to the system state and performance. The CDP requests are similar to those for Datapoints (see Table 7.7), and they are presented in Table 7.8. It should be also noted that the above requests require the following path parameters:

- **assetUrn**: directly printed on an asset (KIS.Device), can be retrieved through the request no. 1 from Table 7.4 or through KIS.MANAGER;
- **calaculatedDatapointDefinition**: CDP name, e.g., kg2lb.

Additionally, the request no. 2 in Table 7.8 can be executed with the following query parameters:

- **from**: an ISO timestamp indicating a lower bound of the required time range;
- **to**: an ISO timestamp indicated an upper bound of the required time range;
- **limit**: a maximum number of required CDP values.

Now, let us proceed to KPI requests available through KIS.API. They are given in Table 7.9, and it is not surprising that they are similar to those presented in Table 7.8. Additionally, they path parameters are given by

- **assetUrn**: directly printed on an asset (KIS.Device), can be retrieved through request no. 1 from Table 7.4 or through KIS.MANAGER;
- **kpiDefinition**: the CDP name, e.g., kg2lb.

Unlike Datapoints and CDP, the KPI request no. 2 (see Table 7.9) has to be executed with the following query parameters:

- **from**: an ISO timestamp indicating a lower bound of the required time range;
- **to**: an ISO timestamp indicate an upper bound of the required time range.

Table 7.8 List of possible requests associated with CDPs

No.	Verb	Path	Action
1	GET	baseUrl/assets/assetUrn/ calculatedDatapointDefinitions	Obtain a list of CDPs
2	GET	baseUrl/assets/assetUrn/ calculatedDatapointDefinition/ calculatedDatapointValues	Obtain CDP values

Table 7.9 List of possible requests associated with KPIs

No.	Verb	Path	Action
1	GET	baseUrl/assets/assetUrn/kpiDefinitions	Obtain a list of KPIs
2	GET	baseUrl/assets/assetUrn/kpiDefinition/ kpiValues	Obtain KPI values

Accessing the list of KPIs

Let us continue with the example presented in Sect. 7.2.2 pertaining to a KIS.BOX associated with two rules. These two rules perform cyclically one after the other. The first one change the KIS.BOX Button 1 operational LED color from black to red while the second one realizes an opposite situation. It is assumed that no KPIs are defined for this KIS.BOX (see Sect. 4.1.2 for a detailed tutorial on KPIs). Thus, let us define the KPI counting the entire time period for which the above mentioned color is red. For that purpose, the following KPI is implemented:

```
y = Round[Sum[If[x == 5, Duration[x], 0]]/1000];
```

where `x` stands for the `button1ColorKpiDuration` Datapoint while the number five signifies the red color (see Table 2.2). Finally, let us assume that this KPI is named `KBButton1Red` while its processing period is set to 15 min. Having the above KPI, let us proceed to performing the request no. 1 in Table 7.9. As a result, the following JSON structure is obtained:

```
[
  {
    "name": "KBButton1Red"
  }
]
```

Accessing KPI values

Let us continue with the above example. Now, the task is to obtain `KBButton1Red` (path parameter `kpiDefinition`) values within the time period defined by the parameters `'from'` and `'to'` given by

```
2022-08-31T08:02:50.046Z
2022-08-31T12:22:50.046Z
```

which, as discussed in Sect. 7.2.2, are formatted according to

```
2022-08-31T08%3A02%3A50.046Z
2022-08-31T12%3A22%3A50.046Z
```

As a result, the following JSON structure is obtained:

```
{
  "to": "2022-08-31T12:22:50.046Z",
  "from": "2022-08-31T08:02:50.046Z",
  "values": [
    447,
    443,
    448,
    447,
    451,
    451,
    448,
    451,
    450,
    453,
    447,
    454
  ]
}
```

It can be easily observed that there are 12 values corresponding to 15-minute processing periods. It is also straightforward to observe that 15 min are equivalent to 900s. Thus, it is evident that all the above-presented values should oscillate around 450s, which is actually the case.

Accessing data from CDPs

The last task of this point concerns obtaining an information about predefined CDPs as well as finding their values. For that purpose, it is assumed that no CDPs are defined. Moreover, the preceding example is continued. Thus, a new CDP is defined according to the approach presented in Sect. 4.1.1 with x equivalent to `button1ColorKpiDuration` Datapoints. The developed CDP aims at bounding the minimum numerical value of x to 3, which represents the green color (see Sect. 2.6). As a result, the following simple implementation is obtained:

$$z = \text{If}[x < 3, 3, x];$$

while CDP itself is named `KBmaxcolorCDP`. Let us start with the request no. 1 in Table 7.8, which pertains to obtaining a list of all available CDPs. As a result of using it, the following JSON structure is arrived at:

```
[
  {
    "name": "KBmaxcolorCDP",
    "datatype": "DOUBLE"
  }
]
```

which contains the existing CDP names as well as their data types. Having the CDP name, let us proceed to obtaining its values. In fact this process is identical to the one presented in Sect. 7.2.2. According to the adjustment performed in the preceding examples, x can have the values representing either the red or the black color, i.e., $x = 5$ or $x = 2$ (cf. Table 2.2). The request No. 2 in Table 7.8 is executed with the query parameter `limit` only, which is equal to 10. As a result, the following JSON structure is obtained, which provides the desired results:

```
[
  {
    "timestamp": "2022-09-02T10:45:22.804Z",
    "value": "5"
  },
  {
    "timestamp": "2022-09-02T10:45:22.335Z",
    "value": "3"
  },
  {
    "timestamp": "2022-09-02T10:45:21.554Z",
    "value": "5"
  },
  {
    "timestamp": "2022-09-02T10:45:20.929Z",
    "value": "3"
  },
  {
    "timestamp": "2022-09-02T10:45:20.054Z",
    "value": "5"
  }
]
```

Table 7.10 List of possible requests associated with rules

No.	Verb	Path	Action
1	GET	baseUrl/rules	Obtain a list of rules
2	GET	baseUrl/rules/ruleId/assetgroupId	Obtain a rule info

7.2.4 Accessing Information About Rules

Rules (cf. Sect. 2.9) constitute the last component which can be accessed through KIS.API. The currently possible requests are provided in Table 7.10.

Additionally, the request no. 2 in Table 7.10 should be executed with the following path parameters:

- ruleId: the rule identification number which can be retrieved through the request no. 1 from Table 7.10.
- assetgroupId: asset group identification.

The parameters that can be accessed through the process of executing these requests are

- name: the name of the rule provided in KIS.MANGER Rule engine;
- enabledAPI: a logical property stating if it is possible to trigger the rule from an external application using KIS.API.

Obtaining information about rules

The objective of this example is to show how to access information about rules. Let us start with the request no. 1 in Table 7.10, which does not require any path or query parameters while its execution results in the following JSON structure:

```
[
  {
    "name": "rblack2red",
    "assetGroupId": 9758,
    "id": "b92080c3-55aa-410e-a417-9efe90637515",
    "enabledAPI": false
  },
  {
    "name": "rred2black",
    "assetGroupId": 9758,
    "id": "62124944-a606-4b80-b906-94d6d5ae8d38",
    "enabledAPI": false
  }
]
```

Contrarily, having `assetGroupId` and `id` signifying the rule, one can obtain the name of the rule and the logical property `enabledAPI`. Indeed, by using them as the path parameters and then executing the request no. 2 in Table 7.10, one can arrive at the following JSON structure:

```
{
  "name": "rblack2red",
  "assetGroupId": 9758,
  "id": "b92080c3-55aa-410e-a417-9efe90637515",
  "enabledAPI": false
}
```

7.2.5 Triggering Rules from External Applications

The objective of this section is to introduce a very important feature of KIS.API, which makes it possible to trigger a rule from an external application. However, as mentioned in Sect. 7.2.4, such an operation is possible for the rules having the `enabledAPI` property set to the logical truth. For example, the rule considered at the end of Sect. 7.2.4 should have the following JSON structure:

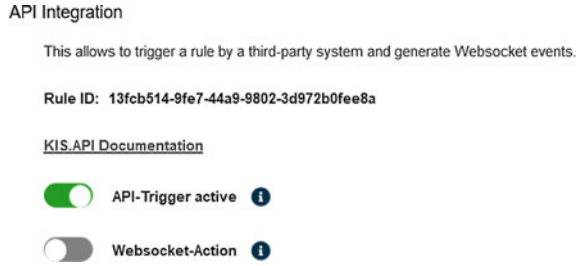
```
{
  "name": "rblack2red",
  "assetGroupId": 9758,
  "id": "b92080c3-55aa-410e-a417-9efe90637515",
  "enabledAPI": true
}
```

Note that the modification of the `enabledAPI` property is possible through KIS.MANGER only. For that purpose, one should use the Main menu → Asset groups and then select an appropriate asset group. Subsequently, by going to Rule engine and selecting the desired rule, one can see the property editor called API integration, which is presented in Fig. 7.3. As can be observed, there is another property, which is called Websocket-Action. However, it will be discussed in Sect. 7.5. Finally, the rule triggering request is described in Table 7.11.

Table 7.11 a requests associated with a rule trigger

No.	Verb	Path	Action
1	POST	<code>baseUrl/rules/RuleId/assetgroupId/Trigger</code>	trigger a rule

Fig. 7.3 Setting enabledAPIproperty



7.3 KIS.API in Practice

The purpose of Sect. 7.2 was to introduce to the reader the essential functionalities concerning KIS.API. All of them were carefully described while their practical usage was explained with the Postman (<https://www.postman.com/>) application. The objective of this section is to provide practical guidelines concerning KIS.API application using some popular software. Indeed, the software selection being used in this section is not accidental. The first candidate is employed widely both in the industry for presenting various kinds of data in tabular order. The second one is commonly used for research, analysis, development and deployment of new practical concepts based on data gathered from a given system. Thus, these two popular software instances are

- Microsoft Excel (<https://www.microsoft.com/>),
- MathWorks Matlab (<https://www.mathworks.com/>).

Both of them have several different and freely-available counterparts, which can provide similar functionalities. Thus, the objective of the subsequent point is to provide a short practical tutorial on feeding MS Excel and Matlab with KIS.ME data. Although the current section is restricted to MS Excel and Matlab, the reader possessing the knowledge about the KIS.ME essential functionalities can integrate it with more advanced and dedicated software. An enterprise resource planning system can be a good example of such software (see, e.g., SAP and its API functionalities at <https://api.sap.com/>).

7.3.1 Feeding MS Excel with KIS.ME Data

Starting with MS Excel 2013 it is possible access any REST API using the so-called power query. Thus, the entire recipe for accessing data from KIS.API boils down the following steps:

1. Select and push the `from Web` power query icon.
2. Provide the URL associated with the desired request.

Convert	Manage Items	Sort	Numeric List
Queries	>		
	List		
	1	Record	
	2	Record	
	3	Record	

Fig. 7.4 Result of a KIS.API request in MS Excel

ABC 123	Column1.name	ABC 123	Column1.assetGroupId	ABC 123	Column1.id	ABC 123	Column1.enabledAPI
1	KBOutHigh		9758	52739801-29c0-4318-9edd-0774fd3d0c44			FALSE
2	rbblack2red		9758	b92080c3-55aa-410e-a417-9efe90637515			FALSE
3	rred2black		9758	62124944-a606-4b80-b906-94d6d5ae8d...			FALSE

Fig. 7.5 Result of a KIS.API request presented as a table

3. Use advanced options to provide appropriate headers, i.e., X-CLIENT-ID and X-API-KEY (see Fig. 7.1).
4. Perform the desired data request.

Obtaining a list of all rules

The objective of this example is to obtain a list of all rules present in KIS.MANAGER Rule engine. According to Sect. 7.2.4, a URL should be defined as follows:

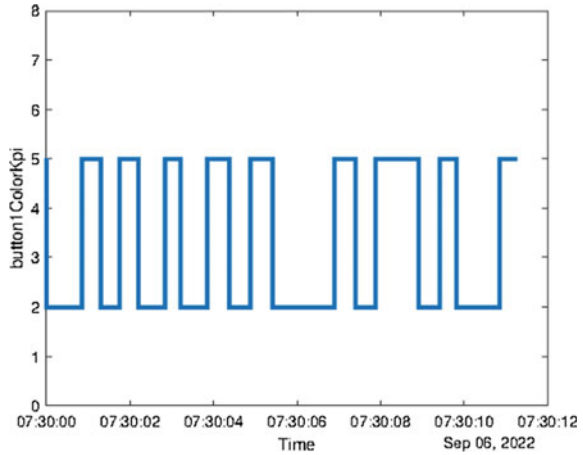
```
baseUrl/rules
```

while the required headers may have the following structure:

```
X-API-KEY 2b84bd4de38b4e92bb7bb283364477e1
X-CLIENT-ID 708ce7cc-fd57-416d-991f-f68704385c22
```

Finally, the desired data request is performed and the obtained result is given in Fig. 7.4. As can be observed, there are three records, which simply correspond to three different rules. To make the obtained result more transparent, the option Convert to Table can be used used, which after suitable expansion yields the view presented in Fig. 7.5.

Fig. 7.6 Result of a KIS.API request in Matlab



7.3.2 Feeding Matlab with KIS.ME Data

The objective of this section is to show how to realise the KIS.API GET request with MATLAB. The entire process boils down to the following steps:

1. Provide the URL associated with the desired request.
2. Define the `weboptions` structure `HeaderFields` containing the returned content type, X-CLIENT-ID and X-API-KEY.
3. Define optional query parameters.
4. Execute `webread` to obtain data associated with the desired request:

```
data=webread(URL,web_options,query_parameters)
```

Accessing Datapoint values

The primary objective of this example is to obtain twenty recent values of the `button1ColorKpi` Datapoint of KIS.BOX defined by a given URN. For that purpose the example introduced in Sect. 7.2 is utilized. Let us start with defining an appropriate URL according to Sect. 7.2.2 (Table 7.7), which can be realized as follows:

```
baseUrl='https://api.kisme.com/kisapi/v1';
assetUrn='urn:rafi:sbox:9c65f93cbcd6';
DPdef='button1ColorKpi';
finalUrl=strcat(baseUrl,'assets/',assetUrn,+'/',DPdef,
'/datapointValues');
```


Subsequently, the `weboptions` structure with `HeaderFields` is defined:

```
opt=weboptions;
content={'Content-Type' 'application/json'};
client={'X-CLIENT-ID' '708ce7cc-fd57-416d-991f-f68704385c22'};
key={'X-API-KEY' '2b84bd4de38b4e92bb7bb283364477e1'};
opt.HeaderFields=[content;client;key];
```

Finally, `webread` can be executed:

```
data=webread(finalUrl,opt,'limit','20');
```

However, the current example, apart from retrieving data, extracts the values and occurrence times of the KIS.API data. Moreover, such a request is repeated every second and the obtained results are suitably visualized. Such a time-driven loop is repeated until a user presses any key. The code realizing all the above mentioned operations is given as follows:

```
clc; clear; close all;
baseUrl='https://api.kiseme.com/kisapi/v1';
assetUrn='urn:rafi:sbox:9c65f93cbed6';
DPdef='button1ColorKpi';
finalUrl=strcat(baseUrl,'assets/',assetUrn,+'/',DPdef,
'/datapointValues');
opt=weboptions;
content={'Content-Type' 'application/json'};
client=
{'X-CLIENT-ID' '708ce7cc-fd57-416d-991f-f68704385c22'};
key={'X-API-KEY' '2b84bd4de38b4e92bb7bb283364477e1'};
opt.HeaderFields=[content;client;key];
h = figure(1);
while isempty(get(h,'CurrentCharacter'));
    data=webread(finalUrl,opt,'limit','20');
    values=[cellfun(@str2num,{data.value})];
    ind=find(values>7);
    values(ind)=[];
    times=datetime({data.timestamp},'InputFormat',
'uuuu-MM-dd' 'T' 'HH:mm:ss.SSSZ','TimeZone','UTC');
    times(ind)=[];
    stairs(times,values,'LineWidth',2);
    xlabel('Time');
    ylabel(DPdef);
    ylim([0 8]);
    pause(1);
end;
```

The obtained results are given in Fig. 7.6. As can be observed, the value of `button1ColorKpi` Datapoints is switched between black (2) and red (5) colors.

Remark 7.1 The example presented in this point corresponds to the so-called polling procedure in which KIS.API requests are repeated every single second. This process is, of course, inefficient as it is executed irrespective of the fact of having new KIS.API data. Indeed, even if there is no new data, the request is still executed. To settle this unappealing phenomenon, KIS.ME provides the so-called websockets, which are discussed in Sect. 7.5.

7.4 Triggering Rules from MATLAB

The objective of this section is to show how to realize a KIS.API POST request with Matlab. For that purpose, an example with triggering the rule is engaged. Generally, the entire process boils down to the following steps:

1. Provide the URL associated with the desired request.
2. Define the `weboptions` structure `HeaderFields` containing the returned content type, X-CLIENT-ID and X-API-KEY.
3. Define optional request parameters.
4. Execute `webread` to obtain the data associated with the desired request:

```
data=webwrite(URL,web_options,request_parameters)
```

Triggering a rule from MATLAB

The objective of this example is to show how to trigger KIS.ME rule from MATLAB. For that purpose, let us define a new rule. This rule has no triggers or conditions defined in KIS.MANGER. It aims at switching the KIS.BOX Button 2 operational LED color to yellow. Thus, there is only one action which performs the above task. Subsequently, API Integration (see Fig. 7.3) is used to set the `enabledAPI` property by activating the API-Trigger active option. The remaining information required to formulate the rule triggering request reduces to collecting

Rule ID: 13fcb514-9fe7-44a9-9802-3d972b0fee8a,
Asset group ID: 9758.

Having the above information, it is possible to formulate the triggering request according to Table 7.11. Finally, the resulting code is given as follows:

```

clc; clear; close all;
baseUrl='https://api.kisme.com/kisapi/v1';
ruleID='13fcb514-9fe7-44a9-9802-3d972b0fee8a';
assetgroupId='9758';
addpath='rules/';
finalUrl=strcat(baseUrl,addpath,ruleID,+ '/' ,assetgroupId,
'/trigger');
opt=weboptions;
content={'Content-Type' 'application/json'};
client=
{'X-CLIENT-ID' '708ce7cc-fd57-416d-991f-f68704385c22'};
key={'X-API-KEY' '2b84bd4de38b4e92bb7bb283364477e1'};
opt.HeaderFields=[content;client;key];
data=webwrite(finalUrl,opt);

```

7.5 Websockets

In spite of an incontestable appeal of the communication strategies presented in the preceding part of this chapter, they frequently suffer from the lack of efficiency. This is particularly the case when there is a need for observing the changes in system behaviour expressed in the evolution of Datapoints associated with KIS.Devices. Indeed, such a problem was already discussed in Sect. 7.3.2. The example considered in the preceding section concerned the so-called polling procedure, in which KIS.API requests are repeated every single second. This process is, of course, inefficient as it is executed irrespective of the fact of having new KIS.API data. Indeed, even if there is no new data, the request is still executed. To settle this unappealing phenomenon, KIS.ME provides the so-called websockets, which are discussed in this section.

7.5.1 Brief Introduction to Websockets

Websocket [4] can be defined as a communication protocol, which permits bidirectional communication between the client and the web server. In a most common case, if a browser visits a web page, then an HTTP request is sent to the associated server. Subsequently, the web server replies by sending the response to the web browser. A similar strategy was realized in the preceding part of this chapter while using a different kind of applications, i.e., POSTMAN, MS Excel and Matlab. Thus, as was

shown in Sect. 7.3.2, if the application wants to receive recently released data, then it must constantly, e.g., every second, send a request to the server. This corresponds to the situation in which the user constantly refreshes the page within the browser. This is also the reason why HTTP is a half duplex, which denotes the fact that the traffic flows in a single direction at a time:

- the client releases a request to the server (one direction);
- the server replies to the request (one direction).

Therefore, it is an obvious fact that it is not an elegant solution, widely called polling. It is defined as a regularly timed synchronous call in which the client releases a request to the server to check if there is any new data available. Such requests are realized using regular time intervals, and the client receives a response irrespective of the availability of the new data. Thus, if there is no new data, then the server replies with a negative response and the connection closes. Hence, polling can be efficient when an exact time interval concerning the release of the new data is known. Unfortunately, as has already been mentioned, KIS.ME is used to model a discrete event system in which the occurrence time of events is not equally distributed over a time horizon. Another communication strategy is called long polling. In this case, the client sends request to the server and opens a connection within some time period. If the server has no new data, then it holds the request and connection open until it has a new data for the client (or a predefined timeout is reached). An alternative communication strategy is called streaming. In this case, the client sends a request to the sever, which maintains an open response that is continually updated. The connection can be open permanently or until a predefined timeout is reached. Note that the server never indicates the completion of the HTTP response, and hence the connection is open continuously.

In order to eliminate the above issues, the concept of websocket was introduced. The websocket is by nature a bidirectional full duplex and single-socket connection. While using it, a single HTTP request is required to open a websocket connection. An appealing property of websocket is that it reuses the same connection in both ways, i.e., client–server and server–client. Owing to the fact that the server can send messages as they are available, the overall latency is reduced. Contrarily to polling, websocket communication is based on a single request, i.e., it is not necessary to wait for another request (along with headers, request parameters, etc.). A concise summary of using the websocket can be formulated as follows:

- it makes real-time communication much more efficient;
- it enables a simpler Web-based communication between the client and server;
- it is a network protocol that enables developing other standard protocols on top of it;
- it overcomes the drawbacks of HTTP with respect to real-time communication.

Similarly to HTTP and HTTPS, the websocket defines two URI schemes, namely, ws and wss, which correspond to standard and encrypted communication between the client and the server. The wss (Websocket Secure) URI scheme corresponds to the websocket connection over transport layer security (TLS). Note that TLS is also

known as SSL (Secure Socket Layer). Thus, the same security mechanism is used as the one employed for HTTPS. This means that while constructing websockets one should use a URL of either the `ws://` or `wss://` form.

7.5.2 *Obtaining a KIS.ME URI and Identifiers*

The objective of this point is to show how to retrieve a URI and an identifier necessary to construct a websocket. The first step on the way towards the above objective is to select the data of interest, which can be the following:

- Datapoints: one can obtain real-time data corresponding to Datapoints or calculated Datapoints (see Sects. 2.5 and 4.1.1) of an asset;
- KPIs: one can obtain data corresponding to KPIs (see Sects. 2.5 and 4.1.1) of an asset, which are calculated every 15 min assuming that the new underlying Datapoint values are available;
- Rules: one can obtain data associated with the triggered rules.

Thus, the required preliminary data is associated with the property `subscribeTo`, which may have the following values:

- datapoint,
- kpi,
- rule.

Subsequently, Datapoints and KPIs require a single or a list of `assetIds` (see Sect. 7.2) while rules require `assetGroupIds`. Finally, all these properties form a JSON data structure, which may look as follows:

```
{
  "assetIds": [
    37,
    66
  ],
  "assetGroupIds": [
    43,
    6
  ],
  "subscribeTo": "datapoint"
}
```

Having the above data along with KIS.API credentials (see Sect.7.1.1), one can perform a POST request according to Table 7.12. As a result, the following JSON structure can be obtained:

Table 7.12 Requests of subscribing/unsubscribing to a websocket

No.	Verb	Path	Action
1	POST	baseUrl/websockets	Subscribe to websocket
1	DEL	baseUrl/websockets	Unsubscribe from a websocket

```
{
  "subscriptionUri": [
    "wss://pubsub.api.kisme.com/
    6d574a1f-4c37-4ab4-9fa6-86fb74a66375"
  ],
  "subscriptionId": "6d574a1f-4c37-4ab4-9fa6-86fb74a66375"
}
```

Analogously, the DEL request will unsubscribe from the websocket.

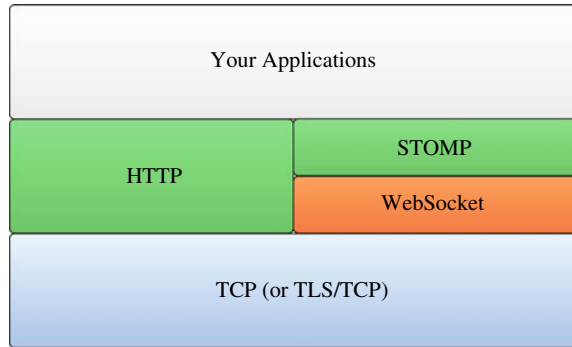
7.5.3 *Brief Introduction to STOMP*

Messaging [4] stands for an architecture associated with sending asynchronous messages between independent components. Such an appealing property makes it possible to develop relatively loosely coupled systems. The crucial components of messaging are the message broker and the client. In particular, the former can perform such actions:

- accepting connections of the clients,
- sending messages to the clients,
- distributing messages among the clients.

Note that the broker can also handle such operations as authorization, message encryption, etc. Thus, if clients are connected to the broker, then they can send messages to the broker as well as receive message distributed by the broker. Such a strategy is called publish/subscribe. Therefore, if a message broker publishes a number of messages, then the client can subscribe to either all or a subset of these messages. STOMP (simple text-oriented messaging protocol) is a good representative example of such a publish/subscribe protocol. Its layering relation with the websocket as well as with other protocols is detailed in Fig. 7.7. STOMP was also employed for the communication purposes within KIS.ME. Indeed, the websocket fits very well to a standard messaging architecture, in which there could be a large volume of potential messages distributed at high rates from the broker to the client. A good example is a client subscribing to Datapoints of an asset (see Sect. 7.5.2). Due to the relatively large number of Datapoints as well as their possible high rate of change,

Fig. 7.7 STOMP over a websocket



receiving messages in real-time as well as with low latency is extremely important for the final performance of the entire application. STOMP is a very simple protocol, which resembles HTTP in its appearance. Each frame consists of a command, headers, etc. STOMP messages can represent any text or binary data. For further information about STOMP, the reader is referred to the STOMP protocol specification [5]. Additionally, STOMP [5] provides the so-called heart-beating mechanism, which can optionally be employed to verify the healthiness of the underlying TCP connection and to ensure that the remote end is still alive and kicking. Generally, it is defined by two integer values, separated by a comma. The first one represents outgoing heart-beats from the sender:

- 0 signifies the fact that it cannot send heart-beats;
- otherwise it is the smallest number of milliseconds between heart-beats that it can guarantee.

The second one represents incoming heart-beats, i.e., what sender would like to obtain:

- 0—stands for the fact that it does not want to receive heart-beats;
- otherwise it is the desired number of milliseconds between heart-beats that it can guarantee.

Note that enabling heart-beating is possible by adding a suitable heart-beating header during the beginning of the STOMP session, i.e., to CONNECT [5].

7.5.4 *Sample Websocket Implementations*

The objective of this section is to provide guidelines for practical implementation of KIS.ME-based websockets. In particular, the NODE.js [1] environment, which employs a widely employed JavaScript (JS) programming language is used. This section is composed of two examples, which aim at

1. reading KIS.ME data using STOMP over websocket,
2. enhancing the above example with a local Web server employed for publishing KIS.ME data.

Moreover, it is assumed that the reader has essential knowledge regarding NODE.js.

Reading KIS.ME data using STOMP over a websocket

Let us start with providing suitable credentials and parameters, which will be located in the `.env` file:

```
SERVER_URL="https://api.kisme.com/kisapi/v1/websockets"
API_KEY="2b84bd4de38b4e92bb7bb283364477e1"
CLIENT_ID="708ce7cc-fd57-416d-991f-f68704385c22"
ASSET_ID="10102"
ASSET_GROUP_ID="9758"
```

For the purpose of further implementations, the following modules are required:

`dotenv`: loads environment variables from the `.env` file into `process.env` structure;

`websocket`: implements the websocket protocol;

`webstomp-client`: provides a STOMP client for Web browsers and NODE.js through websockets;

`axios`: is a promise-based HTTP client for the Web browser and NODE.js.

Note that the application of the above modules is not compulsory and there are several counterparts which can be employed instead. Moreover, their documentation can be easily found at <https://www.npmjs.com>. After such a preliminary step, it is possible to define suitable request headers and options, which are given as follows:

```
const options = {
  method: "POST",
  url: process.env.SERVER_URL,
  headers: {
    "X-API-KEY": process.env.API_KEY,
    "X-CLIENT-ID": process.env.CLIENT_ID,
    "Content-Type": "application/json",
  },
  data: JSON.stringify({
    assetIds: [parseInt(process.env.ASSET_ID)],
    assetGroupIds: [parseInt(process.env.ASSET_GROUP_ID)],
    subscribeTo: "datapoint",
  }),
};
```

while the underlying POST request concerns subscription to a websocket (see Sect. 7.5.2). Thus, the objective is to obtain (cf. `subscribeTo`) the values of

Datapoints and calculated Datapoints of KIS.Devices associated with `assetIds` and `assetGroupIds`. Note that the last property is not compulsory for obtaining Datapoint values. Subsequently, let us assume that both incoming and outgoing heart-beating is set to 1000 ms. Having all the above ingredients, the final code is developed, which is mostly included in the `getSubscriptionId` function:

```

require("dotenv").config();
const WebSocket = require("websocket").w3cwebsocket,
webstomp = require("webstomp-client");
const axios = require("axios");
const heartbeat = 1000;
const getSubscriptionId = () => {
  const options = {
    method: "POST",
    url: process.env.SERVER_URL,
    headers: {
      "X-API-KEY": process.env.API_KEY,
      "X-CLIENT-ID": process.env.CLIENT_ID,
      "Content-Type": "application/json",
    },
    data: JSON.stringify({
      assetIds: [parseInt(process.env.ASSET_ID)],
      assetGroupIds: [parseInt(process.env.ASSET_GROUP_ID)],
      subscribeTo: "datapoint",
    }),
  };
  axios(options)
    .then(function (response) {
      const subscriptionId=response.data.subscriptionId;
      const Server=
response.data.subscriptionUris[0].replace("///","//");
      const socket = new WebSocket(Server, null);
      const stomp = webstomp.over(socket, {
        heartbeat: {incoming: heartbeat, outgoing: heartbeat},
        protocols: ['v12.stomp'],
      });
      stomp.connect(
        {host: Server},
        function () {
          stomp.subscribe(`/topic/${subscriptionId}`,
            function (message) {
              const data = JSON.parse(message.body);
              console.log("Message data",data.info);
            });
        },
      );
    });
};

```

```

    })
    .catch(function (error) {
        console.log(error);
    })
    .finally(function () {
        // always executed
    });
});
};

getSubscriptionId();

```

The example considered is a continuation of the ones exploited in this chapter for which the KIS.BOX (`assetIds=10102`) Button 1 operational LED color switches between red and black. The above KIS.BOX has also associated calculated Datapoint and KPI. After running the above code, one can observe that `message.body` contains the JSON structure, which may look as follows:

```

{"jsonType":"centersightEvent",
 "type":"datapointValuesReceived",
 "nodeId":10102,"timestamp":"2022-10-06T10:47:47.833Z",
 "info":{"key":"button1Color","value":"#000000",
 "timestamp":"2022-10-06T10:47:47.833Z"}}

```

The above JSON structure is self-explained and it can be easily observed that it contains the `info` property, which covers another JSON structure involving

- key: the name of the (calculated) Datapoint,
- value: the value of the (calculated) Datapoint,
- timestamp: the timestamp associated with the value of the (calculated) Datapoint,

and hence, this structure is directly displayed in the console. Note that the above code can be easily adapted to the remaining possible settings of `subscribeTo`, i.e., `kpi` and `rule`. However, such an implementation is left out to be featured an exercise listed in the last section of this chapter.

> Getting information about the rules

Contrarily, to KPIs and Datapoints, rules are directly associated with asset groups. Indeed, there are designed within each asset group. This implies the necessity of a reduced subscription data:

```

{
  assetGroupIds: [parseInt(process.env.ASSET_GROUP_ID)],
  subscribeTo: "rule",
}

```

while `assetIds` is excluded.

Publishing KIS.ME data with a local Web server

The objective of this example is to extend the proceeding one in such a way as to provide the following functionalities:

- feeding the selected Datapoint data to another bi-directional third party API,
- a frontend displaying the selected Datapoint data obtained from the above API.

Let us start with selecting an API. Since the purpose of the example is to collect the data in the form of a JSON structure containing three properties (key, value and timestamp) the list of possible candidates is rather long. Thus, due to relative usage simplicity, the Pusher API was selected (<https://pusher.com/>). As was the case with KIS.API, the first step is to register with Pusher and then collect the list of credentials (App keys). A sample list of Pusher credentials is given as follows:

```

app_id = "1482901"
key = "8e17772867a31d055131"
secret = "b2632afdbd8419d40bf1"
cluster = "eu"

```

Thus, let us extend the `.env` file with the above data, which yields

```

SERVER_URL="https://api.kisme.com/kisapi/v1"
API_KEY="2b84bd4de38b4e92bb7bb283364477e1"
CLIENT_ID="708ce7cc-fd57-416d-991f-f68704385c22"
ASSET_ID="10102"
ASSET_GROUP_ID="9758"
app_id = "1482901"
key = "8e17772867a31d055131"
secret = "b2632afdbd8419d40bf1"
cluster = "eu"

```

Having the above information, let us simply extend the code from the previous example with a list of commands creating a Pusher instance:

```
const Pusher = require("pusher");
const pusher = new Pusher({
  appId: "1482901",
  key: process.env.key,
  secret: process.env.secret,
  cluster: process.env.cluster,
});
```

Now let us assume that the Datapoint of interest is called `button1ColorKpiDuration` (see Appendix. B for its description). Thus, the transfer of Datapoint values to the Pusher API reduces to the following:

```
if (data.info.key=="button1ColorKpiDuration")
  pusher.trigger("b1ColorKpiDuration", "b1ColorKpiDuration",
    {
      value: JSON.stringify(data.info),
    });
```

where `b1ColorKpiDuration` signifies both the so-called channel and event (see <https://www.npmjs.com/package/pusher> for a detailed explanation). The preparation of the backend concludes with including the code for the local Web server. For that purpose the `express` module is used, which can be simply characterized as a lightweight NODE.js Web server. The entire code reduces to adding the following lines:

```
const express = require("express");
const app = express();
app.use(express.static(__dirname + '/public'));
app.listen(3000, () => {
  console.log("Server running on: http://localhost:3000/");
});
```

which are responsible for creating an `express` Web server that will run on port 3000 and will communicate with server static files located in the `public` directory. Finally, the entire backend code can be implemented as follows:

```
require("dotenv").config();
const WebSocket = require("websocket").w3cwebsocket,
webstomp = require("webstomp-client");
const axios = require("axios");
const Pusher = require("pusher");
const pusher = new Pusher({
  appId: "1482901",
  key: process.env.key,
  secret: process.env.secret,
```

```

    cluster: process.env.cluster,
  });
const heartbeat = 1000;
const getSubscriptionId = () => {
  const options = {
    method: "POST",
    url: process.env.SERVER_URL,
    headers: {
      "X-API-KEY": process.env.API_KEY,
      "X-CLIENT-ID": process.env.CLIENT_ID,
      "Content-Type": "application/json",
    },
    data: JSON.stringify({
      assetIds: [parseInt(process.env.ASSET_ID)],
      assetGroupIds: [parseInt(process.env.ASSET_GROUP_ID)],
      subscribeTo: "datapoint",
    }),
  };
  axios(options)
    .then(function (response) {
      const subscriptionId=response.data.subscriptionId;
      const Server=
        response.data.subscriptionUris[0].replace("///", "//");
      const socket = new WebSocket(Server, null);
      const stomp = webstomp.over(socket, {
        heartbeat: {incoming: heartbeat, outgoing: heartbeat},
        protocols: ['v12.stomp'],
      });
      stomp.connect(
        {host: Server},
        function () {
          stomp.subscribe(`/topic/${subscriptionId}`,
            function (message) {
              const data = JSON.parse(message.body);
              if (data.info.key=="button1ColorKpiDuration")
                pusher.trigger("b1ColorKpiDuration",
                  "b1ColorKpiDuration",
                  {
                    value: JSON.stringify(data.info),
                  });
            });
        });
    },
    );
  })
  .catch(function (error) {
    console.log(error);
  });
}

```

```

    })
    .finally(function () {
      // always executed
    });
  });
getSubscriptionId();
const express = require("express");
const app = express();
app.use(express.static(__dirname + '/public'));
app.listen(3000, () => {
  console.log("Server running on: http://localhost:3000/");
});

```

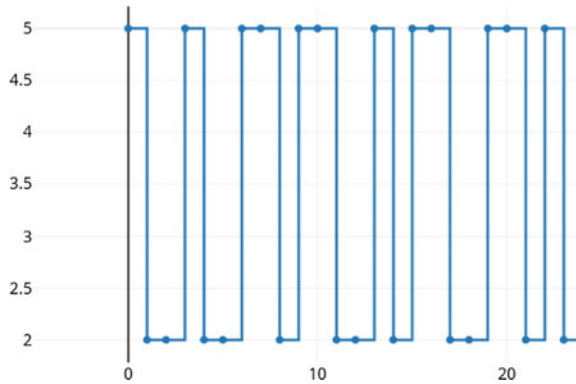
Let us proceed to the frontend development by creating the `public` directory and the `index.html` file inside it. The presentation of Datapoint values will be reduced to showing its consecutive values in the form of a plot. For that purpose the well known `plotly` is employed. Thus, the entire frontend reduces to the following code:

```

<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html;
    charset=UTF-8" />
  <script src="https://cdn.plot.ly/plotly-latest.min.js">
  </script>
  <script src="https://js.pusher.com/7.2.0/pusher.min.js">
  </script>
</head>
<body>
  <div id="chart"></div>
<script>
  Pusher.logToConsole = true;
  var my_plot = {
    y: [],
    mode: 'lines+markers',
    name: 'hv',
    line: {shape: 'hv'},
    type: 'scatter',
  };
  Plotly.newPlot('chart', [my_plot]);
  const pusher = new Pusher(
    "8e1772867a31d055131", // Replace with your 'key'
    { cluster: "eu", }
  );
  const channel = pusher.subscribe("b1ColorKpiDuration");

```

Fig. 7.8 Plotting datapoint values with a local Web server



```

var cnt=0;
var width_graph_window=23;
channel.bind("b1ColorKpiDuration", (data) => {
  const obj=JSON.parse(data.value);
  Plotly.extendTraces('chart', {y:[[obj.value]]}, [0]);
  cnt++;
  if( cnt > width_graph_window)
    Plotly.relayout('chart', {
      xaxis:
        { range: [cnt-width_graph_window, cnt] }
    });
});
</script>
</body>
</html>

```

The main part of the code starts with creating an empty plot located in the `chart` section of the HTML document (`newPlot`). Subsequently, a new Pusher instance is created with the above-defined credentials named `key` and `cluster`. This enables forming a new channel `b1ColorKpiDuration`. Finally, the `channel.bind` command is responsible for receiving cyclically arriving Datapoint values. It also invokes the `extendTrace` command, which updates the existing plot with the new data. Note that the plot is restricted to presenting 23 most up to date values, which requires appropriate scaling realized with the `relayout` command. The graphical result obtained after running the entire application, i.e. the one presented in the browser, is given in Fig. 7.8.

7.6 Training Exercises

7.1 Obtaining a list of assets

1. Obtain a JSON structure containing all KIS.Devices which are at your disposal.
2. Find all assets which are on-line.

7.2 Obtaining a list of asset groups

1. Obtain a JSON structure containing all asset groups.
2. Modify the name and description of a selected asset group.

7.3 Obtaining a list of users

1. Obtain a JSON structure containing all users and determine their `accountNumber`.
2. Obtain a user and determine the above JSON structure once again.
3. Delete the added user.

7.4 Obtaining a list of Datapoints and CDPs

1. Select a KIS.Device and determine its URN.
2. Obtain a JSON structure containing a list of all Datapoints.
3. Determine a list of CDPs.

7.5 Obtaining values of Datapoints and CDPs

1. Choose a KIS.Device and write a set of rules transferring automatically and cyclically its selected operational LED within a state-space: red, yellow, green.
2. Obtain a JSON structure containing a list of 10 recent values of Datapoints corresponding to the numerical values of the operational LED's colors.
3. Prepare CDP converting the numerical values of the operational LED's colors in such a way so that red corresponds to 0, yellow is signified by 1, while green yields 2.
4. Obtain a JSON structure containing a list of 10 recent values of the above CDP.

7.6 Obtaining values of KPIs

1. Continue Exercise 7.5 by implementing KPIs calculating mean times of each state, i.e., red, yellow and green.
2. Obtain a JSON structure containing all KPIs.
3. Obtain a JSON structure containing a list of 10 recent values of the above KPIs.

7.7 Obtaining information about rules

1. Continue Exercise 7.5 and display the information about rules used for color state transitions.
2. Import the information about the rules to MS Excel or any compatible software.

7.8 Triggering rules from an external application

1. Select a KIS.LIGHT.
2. Prepare two rules with API-Trigger active (see Fig. 7.3):
 - GoRed: with an action setting the KIS.LIGHT operational LED color to red;
 - GoGreen: with an action setting the KIS.LIGHT operational LED color to red;
3. Develop a Matlab (you can also use OCTAVE or Python) program, which will trigger these rules every second one after the other.

7.9 Websocket implementation

1. Continue Exercise 7.5 and develop websocket-based application according to the scheme presented in Sect. 7.5.4.
2. Use the above-developed software to get information about the triggered rules and KPI calculation. Hint: use the `subscribeTo` property.

7.10 Websocket implementation

1. Continue Exercise 7.9 and extend it according to Sect. 7.5.4 in order to get a local web server employed for presenting selected Datapoint values.
2. On the frontend side (`index.html`) implement a functionality calculating the total accumulated time of the red color state, i.e., the overall time in which the operational LED color is red.

7.7 Concluding Remarks

The aim of this section was to provide an overview of KIS.API functionalities. It was demonstrated that the software provides an effective way for communicating with external applications. In particular, the chapter started with KIS.API user registration and goes through KIS.API essential functionalities. These functionalities are strictly linked with the content of the preceding chapter. Indeed, it is shown how to prepare request for accessing the information about assets, asset groups, users, Datapoints, CDPs, KPIs as well as the rules. The crucial functionality, which makes KIS.API a fully bidirectional framework is the ability of triggering the rules from external applications. In particular, it was shown how to trigger such rules from Matlab, which is one of the most popular development tools in modern engineering. The last part of the chapter was devoted to the development of efficient websocket-based communication framework, which utilizes STOMP messaging architecture. Indeed, the websocket is an excellent for providing an efficient asynchronous bidirectional communication. It was also demonstrated how to prepare a local Web server for publishing Datapoint values. This crucially example clearly shows that with KIS.API there is an infinite spectrum of possible external extensions of KIS.ME. Finally, the chapter is summarized with a set of training exercises, which verify KIS.API-oriented skills.

References

1. F. Doglio, *Pro REST API Development with Node.js* (Apress, Berkeley, 2015)
2. A. Tamboli, *Build Your Own IoT Platform* (Springer, Berlin, 2019)
3. S. Patni, *Pro RESTful APIs* (Springer, Berlin, 2017)
4. V. Wang, F. Salim, P. Moskovits, *The Definitive Guide to HTML5 WebSocket*, vol. 1 (Springer, New York, 2013)
5. Stomp. <http://stomp.github.io/stomp-specification-1.2.html#Heart-beating>. Accessed: 10 June 2022

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Appendix A

KIS.ME Commands and Their Sample Applications

The objective of this appendix is to provide a comprehensive overview of the KIS.ME commands, which can be utilized while implementing KPIs and CDPs. They are divided into five groups, and their application range is given in Table A.1. This clearly means that Aggregations and Intervals can be used while implementing KPIs only. The list of numeric commands contains essential operations like Plus, Time, Power, Round and Abs, which are very basic, and hence their description is omitted. Similarly, the comparison commands do not require special explanations, either.

A.1 Aggregations

See Table A.2.

$$y = \text{Count}[x] \tag{A.1}$$

Table A.1 KIS.ME commands and their application scope

Group name	KPI	CDPs
Aggregations	YES	NO
Intervals	YES	NO
Numeric	YES	YES
Comparison	YES	YES
Miscellaneous	YES	YES

Table A.2 Count: input and output arguments

Argument	Type	Description
x	Double/long/Boolean	Processed variable
y	Double/long	Number of entries

Table A.3 Count: KPI parameters

Parameter	Value
Definition name	GetCount
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpi as variable x

Illustrative example: The worker assembles products on a workstation. The fact of a well-assembled product is signaled with a blink of a green light of the KIS.LIGHT operational LED. Contrarily, the fact of an inappropriately assembled product is signaled with the KIS.LIGHT operational LED flashing in red. Note that the idle state color of the KIS.LIGHT operational LED is black. The objective is to calculate the percentage number of well-assembled products within a selected processing period. Hint: See numerical values of the red and green colors in Table 2.2.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.3:

Step 2. Output calculation:

```
xg=If[x==3,1,0];
yg=Count[Filter[xg>0]];
xr=If[x==5,1,0];
yr=Count[Filter[xr>0]];
total=yg+yr;
y=yg*100/total;
```

See Table A.4.

$$y = \mathbf{FallingEdge}[x, n], \quad y = \mathbf{FallingEdge}[x] \quad (\text{A.2})$$

Table A.4 FallingEdge: input and output arguments

Argument	Type	Description
x	Double/long/Boolean	Processed variable
n	Double/long	Threshold value for x of either Double/Long type only
		It is the value that needs to be crossed to detect the falling edge, e.g., for $n = 3$, the falling edge will be detected iff x falls below 3
y	Double/long	Number of falling edges

Table A.5 FallingEdge: KPI parameters

Parameter	Value
Definition name	GetFallingEdge
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpi as variable x

Illustrative example: Let us consider a worker performing tasks on a single workstation. If a product assembly is accomplished, then the worker presses KIS.BOX Button 1. The KIS.BOX Button 1 operational LED is normally illuminating in green (numerical value 3), but there is a rule which, after pressing KIS.BOX Button 1, changes this color into black (numerical value 0), and then again to green. The objective is to calculate the number of assembled products within a selected processing period.
 Solution:

Step 1. Define a new KPI and its parameters according to Table A.5:

Step 2. Output calculation:

```
y=FallingEdge[x,3];
```

See Table A.6.

$$y = \mathbf{Max}[x_1, \dots, x_n] \tag{A.3}$$

Table A.6 Max: input and output arguments

Argument	Type	Description
x_1, \dots, x_n	Double/long	n Variables or constants
y	Double/long	Maximum of all x_i , $i = 1, \dots, n$

Table A.7 Max: KPI parameters

Parameter	Value
Definition name	GetMax
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	button1ColorKpiDuration as variable x

Illustrative example: Let us consider an assembly station with one worker, which is associated with KIS.BOX Button 1. Let us assume that there are suitable rules, and hence, if the worker is in an idle state, then the KIS.BOX Button 1 operational LED is black (it is not lighting, the black color numerical value is equal to 2). Calculate the range (in minutes) of idle state periods, i.e., the difference between the maximum and minimum idle state cycles, within a selected processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.7:

Step 2. Output calculation:

```
idle=If[x==2,Duration[x],0]/60000;
z=Filter[idle>0];
y=Max[z,z]-Min[z,z];
```

Table A.8 Mean: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
y	Double/long	Mean value of the processed variable

Table A.9 Mean: KPI parameters

Parameter	Value
Definition name	Get Mean
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

See Table A.8.

$$y = \mathbf{Mean}[x] \tag{A.4}$$

Illustrative example: Let us imagine that the KIS.LIGHT operational LED color is used for indicating the state of a worker. In particular, the magenta color (numerical value equal to 4) signifies the fact that the worker is performing some tasks while the black one expresses an idle state. The objective is to calculate an average working time (in minutes) within a selected processing period.

Solution:

- Step 1. Define a new KPI and its parameters according to Table A.9:
- Step 2. Output calculation:

```
working=If[x==4,Duration[x],0]/60000;
y=Mean[Filter[working>0]];
```

See Table A.10.

$$y = \mathbf{Min}[x, \dots, x_n] \tag{A.5}$$

Table A.10 Min: Input and output arguments

Argument	Type	Description
x_1, \dots, x_n	Double/long	n Variables or constants
y	Double/long	minimum of all $x_i, i = 1, \dots, n$

Table A.11 Min: KPI parameters

Parameter	Value
Definition name	GetMin
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	Button1ColorKpiDuration as variable x

Illustrative example: Let us consider an assembly station with one worker, which is associated with KIS.BOX Button 1. Let us assume that there are suitable rules, and hence, if the worker is in an idle state, then the KIS.BOX Button 1 operational LED is black (it is not lighting, the black color numerical value is equal to 2). Calculate the minimum (in minutes) idle state period within a selected processing cycle.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.11:

Step 2. Output calculation:

```
idle=if[x==2,Duration[x],0];
z=Filter[idle>0];
y=Min[z,z]
```

See Table A.12.

$$y = \text{Percentile}[x, n] \quad (\text{A.6})$$

Illustrative example: Let us imagine that the KIS.LIGHT operational LED color is used for indicating the state of a worker. In particular, the magenta color (numerical value equal to 4) signifies the fact that the worker is performing some tasks while

Table A.12 Percentile: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
n	Double/long	Number indicating the n -th percentile
y	Double/long	n -th Percentile of the processed variable

Table A.13 Percentile: KPI parameters

Parameter	Value
Definition name	Get percentile
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

the black one expresses an idle state. The objective is to calculate a median working time (in minutes) within a selected processing period. Hint: The median value is the 50th percentile.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.13:

Step 2. Output calculation:

```
working=If[x==4,Duration[x],0]/60000;
y=Percentile[Filter[working>0],50];
```

See Table A.14.

$$y = \text{RisingEdge}[x, n] \quad \text{or} \quad y = \text{RisingEdge}[x] \quad (\text{A.7})$$

Illustrative example: A set of parts is supplied to a single workstation by a conveyor belt. Such a situation is indicated by feeding KIS.LIGHT digital input 1 with a false–true–false sequence. The objective is to calculate the number of sets of parts supplied within a selected processing period.

Table A.14 RisingEdge: Input and output arguments

Argument	Type	Description
x	Double/long or Boolean	Numerical or Boolean variable
n	Double/long	Threshold value for x of either double/long type only It is the value that needs to be crossed to detect the rising edge, e.g. for $n = 3$, the rising edge will be detected iff x exceeds 3
y	Double/long	Number of rising edges

Table A.15 RisingEdge: KPI paramters

Parameter	Value
Definition name	GetRisingEdge
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	input1Status as variable x

Table A.16 Stdev: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
y	Double/long	Standard deviation of the processed variable

Solution:

Step 1. Define a new KPI and its parameters according to Table A.15.

Step 2. Output calculation:

```
y=RisingEdge[x];
```

See Table A.16.

$$y = \text{Stdev}[x] \quad (\text{A.8})$$

Table A.17 Stdev: KPI parameters

Parameter	Value
Definition name	Get Stdev
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

Table A.18 Sum: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
y	Double/long	Sum of x entries

Illustrative example: Let us imagine that the KIS.LIGHT operational LED color is used for indicating the state of a worker. In particular, the magenta color (numerical value equal to 4) signifies the fact that the worker is performing some tasks while the black one expresses an idle state. The objective is to calculate a standard deviation of the working time (in minutes) within a selected processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.17:

Step 2. Output calculation:

```
working=If[x==4,Duration[x],0]/60000;
y=Stdev[Filter[working>0]];
```

See Table A.18.

$$y = \text{Sum}[x] \tag{A.9}$$

Illustrative example: Let us imagine that KIS.LIGHT operational LED color is used for indicating the state of a worker. In particular, the magenta color (numerical value equal to 4) signifies the fact that the worker is performing some tasks while the black one expresses an idle state. The objective is to calculate the total working time (in minutes) within a selected processing period.

Table A.19 Sum: KPI parameters

Parameter	Value
Definition name	GetSum
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input Datapoints	led1ColorKpiDuration as variable x

Solution:

Step 1: Define a new KPI and its parameters according to Table A.19

Step 2: Output calculation

$$y = \text{Sum}[\text{If}[x==4, \text{Duration}[x], 0]] / 60000;$$

A.2 Intervals

See Table A.20.

$$y = \mathbf{End}[x], y = \mathbf{Start}[x] \quad (\text{A.10})$$

Illustrative example: A worker performs assembly tasks, which are associated with the following KIS.LIGHT operational LED colors:

- Green: battery cell mounting,
- Red: battery controller mounting,
- Black: an idle state of the worker.

Table A.20 Start/End: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
y	Double/long	Start/end timestamp of all variable states

Table A.21 Count: KPI paramters

Parameter	Value
Definition name	GetEndStart
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

Table A.22 Duration: input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
y	Double/long	Durations of individual variable states

Calculate the total mounting time (in minutes) within a selected processing period.
 Solution:

Step 1. Define a new KPI and its parameters according to Table A.21:

Step 2. Output calculation:

```
dx=If [x==3 | x==5, End[x]-Start[x], 0];
y=Sum [dx] / 60000;
```

See Table A.22.

$$y = \mathbf{Duration}[x] \tag{A.11}$$

Illustrative example: A worker performs assembly tasks associated with the following KIS.LIGHT operational LED colors:

- Green: a battery cell mounting,
- Red: a battery controller mounting,
- Black: an idle state of the worker.

Calculate the ratio between the total cell and controller mounting times within a selected processing period.

Table A.23 Duration: KPI parameters

Parameter	Value
Definition name	GetDuration
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

Table A.24 Interval: Output arguments

Argument	Type	Description
y	Double/long/Boolean	KPI processing period

Solution:

Step 1. Define a new KPI and its parameters according to Table A.23:

Step 2. Output calculation:

```
dg=Sum[If[x==3,Duration[x],0]];
dr=Sum[If[x==5,Duration[x],0]];
y=dg/dr;
```

See Table A.24.

$$y = \text{Interval[]} \quad (\text{A.12})$$

Illustrative example: Let us imagine that the KIS.LIGHT operational LED color is used for indicating the state of a worker. In particular, the magenta color (numerical value equal to 4) signifies the fact that the worker is performing some tasks while the black one expresses an idle state. The objective is to find the percentage of the working state within a selected processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.25:

Step 2. Output calculation:

```
y=Sum[If[x==4,Duration[x],0]]/Interval[]*100;
```

Table A.25 Interval: KPI paramters

Parameter	Value
Definition name	GetInterval
Published as	KPI
Processing period	Select a desried one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

A.3 Numeric

The list of numeric commands contains essential operations like Plus, Time, Power, Round and Abs, which are very basic, and hence their description is omitted.

See Table A.26.

$$y = \mathbf{Mod}[x, n] \tag{A.13}$$

Illustrative example: Let us imagine that the tasks realized by two machines are indicated via the KIS.LIGHT operational LED that illuminates in task-oriented colors. In particular, the colors with even numeric values (cf. Table 2.2) are associated with the tasks of the first machine. Contrarily, the jobs realized by the second machine are identified by the colors with odd numeric values. The objective is to calculate the ratio between the total number of tasks realized by the machines. Hint: Filter all values, which are numeric values indicated in Table 2.2.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.27:

Step 2. Output calculation:

Table A.26 Mod: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
n	Double/long	Divider
y	Double/long	Division remainder $y = x - n \left\lfloor \frac{x}{n} \right\rfloor$ where $\lfloor \cdot \rfloor$ signifies the floor function

Table A.27 Mod: KPI parameters

Parameter	Value
Definition name	GetMod
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input Datapoints	led1ColorKpi as variable x

Table A.28 Bit: Input and output arguments

Argument	Type	Description
x	Double/long	Processed variable
n	Double/long	Bit position
y	Double/long	n -th Bit of x

```
dx=Filter [x<8];
even=Sum [If [Mod [x, 2]==0, 1, 0]];
odd=Sum [If [Mod [x, 2]==1, 1, 0]];
y=even/odd;
```

A.4 Miscellaneous

See Table A.28.

$$y = \mathbf{Bit}[x, n] \quad (\text{A.14})$$

Illustrative example: The colors of the KIS.LIGHT operational LED are associated with the tasks realized by two groups of workers. The third bit of the numerical color values associated with the first group is equal to 1. Contrarily, the same bit is equal to 0 for the second group of the color numerical values. The objective is to implement a CDP which will return 1 for the tasks associated with the first group and 0 otherwise. Solution:

Step 1. Define a new CDP and its parameters according to Table A.29:

Step 2. Output calculation:

Table A.29 Bit: CDP parameters

Parameter	Value
Definition name	Get Bit
Published as	Calculated Datapoint
Input Datapoints	led1ColorKpi as variable x

Table A.30 Counter: CDP parameters

Parameter	Value
Definition name	Get Bit
Published as	Calculated Datapoint
Input Datapoints	led1ColorKpiDuration as variable x

```
xf=Filter[x<8];
y=Bit[xf,3];
```

$$y = \mathbf{Counter}[x, b] \tag{A.15}$$

Argument	Type	Description
x	Long/double	Possibly varying value (x_c, x_p signify x at $c > p$ time instances)
b_c	Long/double	Possibly time-varying bias
y	Long/double	If $x_c \geq x_p$ then $y = (x_c - x_p)$, else $y = b_c + (x_c - x_p)$

Illustrative example: The objective is to calculate the difference between consecutive numerical values of the KIS.LIGHT operational LED colors (Table A.30.).

Solution:

Step 1. Define a new CDP according to Table A.30:

Step 2. Output calculation:

```
y=Counter[x,0];
```

Table A.31 If: Input and output arguments

Argument	Type	Description
x	Boolean	Processed decision variable
a	Double/long	Input variable
b	Double/long	Input variable
y	Double/long	If x is true, then $y = a$, else $y = b$

Table A.32 If: CDP parameters

Parameter	Value
Definition name	Get If
Published as	Calculated datapoint
Input datapoints	button1ColorKpi as variable x
Input datapoints	button2ColorKpi as variable y

See Table [A.31](#).

$$y = \mathbf{If}[x, a, b] \quad (\text{A.16})$$

Illustrative example: The colors of the KIS.BOX Button operational LEDs are associated with the tasks realized by two group of workers. Implement a CDP calculating the current minimum of the numerical values associated with these operational LEDs. Solution:

Step 1. Define a new CDP and its parameters according to Table [A.32](#):

Step 2. Output calculation:

$$z = \mathbf{If}[x < y, x, y];$$

See Table [A.33](#).

$$y = \mathbf{Filter}[c] \quad (\text{A.17})$$

Table A.33 If: Input and output arguments

Argument	Type	Description
<i>c</i>	Boolean	Logical condition acting on variable <i>x</i> , e.g., $x > 0$
<i>y</i>	Double/long	Values of <i>x</i> satisfying <i>c</i>

Table A.34 Filter: KPI parameters

Parameter	Value
Definition name	GetFilter
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable <i>x</i>

Illustrative example: The colors of the KIS.LIGHT operational LED indicate the tasks realized by an assembly station. Calculate the total time of tasks associated with the red color which are longer than two minutes. Perform calculation within a desired processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table A.34:

Step 2. Output calculation:

```
d=If[x==5,Duration[x],0];
y=Sum[Filter[d>2*60000]];
```

Appendix B

KIS.ME Datapoints and Their Sample Applications

button1ColorKPIDuration (B.1)

Description: A long type Datapoint associated with the KIS.BOX Button 1 operational LED. It contains information about the evolution of the color states of the button as well as their durations. A similar Datapoint is associated with the KIS.BOX Button 2 operational LED and is called `button2ColorKPIDuration`.

Illustrative examples:

KPI: There are two tasks associated with two colors of the KIS.BOX Button 1 operational LED, i.e., blue (numerical value 0) and green (numerical value 3). The objective is to obtain the durations of states corresponding to these colors and find the maximum one (in minutes) within a selected processing period.

CDP: There are two workers associated with KIS.BOX Buttons 1 and 2. Their idle state is signified by the green color of the respective operational LED. The objective is to implement an indicator routine, which will return 1 when both workers are in the idle state and 0 otherwise.

KPI solution:

- Step 1. Define a new KPI and its parameters according to Table B.1:
- Step 2. Output definition:

```
blue = If[x==0,Duration[x],0];
green = If[x==3,Duration[x],0];
y = Round[Max[blue,green]/60000];
```

Table B.1 Maximum duration KPI parameters

Parameter	Value
Definition name	MaxTime
Published as	KPI
Processing period	Select a desired on
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	button1ColorKPIDuration as variable x

Table B.2 Idle state CPD parameters

Parameter	Value
Definition name	IdleTime
Published as	Calculated Datapoint
Input datapoints	button1ColorKPIDuration as variable $x1$ textttbutton2ColorKPIDuration as variable $x2$

CDP solution:

Step 1. Define a new CDP and its parameters according to Table B.2

Step 2. Output definition:

```
idle=If[x1==3 && x2==3,1,0]
```

button1ColorKPI

(B.2)

Description: A long type Datapoint associated with the KIS.BOX Button 1 operational LED. It contains information about the evolution of the color states of the button. A similar Datapoint is associated with the KIS.BOX Button 2 operational LED and is called button2ColorKPI.

Illustrative example:

CDP: It is possible that during the transient between the color states the value of button1ColorKPI may be outside the admissible color range (cf. Table 2.2). The objective is to implement a routine eliminating such values.

Table B.3 Admissible colors CDP parameters

Parameter	Value
Definition name	Filtration
Published as	Calculated datapoint
Input datapoints	button1ColorKpi as variable x

Table B.4 Minimum task number KPI parameters

Parameter	Value
Definition name	MinTasks
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input calculated datapoint	Filtration as variable x

KPI: Use the results provided by the above CDP to calculate the number of realized tasks, which are associated with two colors of the KIS.BOX operational LED. These are blue (numerical value 0) and green (numerical value 3). Finally, determine the minimum number of tasks realized by these two groups within a desired processing period.

CDP solution:

- Step 1. Define a new CDP and its parameters according to Table B.3:
- Step 2. Output definition:

```
res=Filter[x<=7];
```

KPI solution:

- Step 1. Define a new KPI and its parameters according to Table B.4:
- Step 2. Output KPI definition

```
:
blue=Sum[If[x == 0, 1, 0]];
green=Sum[If[x == 3, 1, 0]];
y = Min[blue,green];
```

Table B.5 Minimum number of tasks KPI parameters

Parameter	Value
Definition name	MinPressed
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input Datapoints	button1Pressed as x1
	button2Pressed as x2

button1Pressed

(B.3)

Description: A Boolean type Datapoint associated with KIS.BOX Button 1. It contains information about the states related to pressing this button, which are equivalent to the logical truth. A similar Datapoint is associated with KIS.BOX Button 2 and is called `button2Pressed`.

Illustrative example: There are two workers which are associated with KIS.BOX Buttons 1 and 2. Each of them presses the respective button after realizing a given task. The objective is to implement a routine calculating the minimum number of tasks realized by either worker within a selected processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table B.5:

Step 2. Output definition:

```
Total=Min[Sum[If[x1, 1, 0]], Sum[If[x2, 1, 0]]];
```

isOnline

(B.4)

Description: A Boolean type Datapoint associated with a KIS.Device, which contains information about its online states, i.e., connection to WiFi, along with their durations.

Table B.6 Total online duration KPI parameters

Parameter	Value
Definition name	OnlineTime
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoint	isOnline as variable <i>x</i>

Illustrative example: Calculate the total duration (in minutes) of online states of KIS.Device within a desired processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table B.6:

Step 2. Output KPI definition:

```
y = Round[Sum[If[x, Duration[x], 0]]/60000];
```

led1ColorKPIDuration

(B.5)

Description: A long type Datapoint associated with the KIS.LIGHT operational LED. It contains information about the evolution of the color states of this LED as well as their durations.

Illustrative example: A machine performs an assembly process, which is signified by the blue color (numerical value 0) of the KIS.LIGHT operational LED. If the assembly is completed, then the color is changed into red (numerical value 5). The objective is to calculate a median duration (in minutes) of realizing the tasks indicated by the blue color within a selected processing period. Hint.: The median value corresponds to the 50th percentile.

Solution:

Step 1. Define the new KPI and its parameters according to Table B.7:

Step 2. Output KPI definition:

```
colorBlue=If[x==0,Duration[x],0]/60000;
y=Percentile[Filter[colorBlue>0],50];
```


Table B.7 Median duration KPI parameters

Parameter	Value
Definition name	Percentile
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpiDuration as variable x

Table B.8 Even and odd tasks KPI parameters

Parameter	Value
Definition name	Ratio
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input datapoints	led1ColorKpi as variable x

ledColorKPI

(B.6)

Description: A long type Datapoint associated with the KIS.LIGHT operational LED. It contains information about the evolution of the color states of this LED.

Illustrative example: The KIS.LIGHT operational LED indicates tasks realized by two workers. In particular, the colors with even numerical values are assigned to the tasks realized by the first worker while those with odd ones are associated with the second one. The objective is to calculate the ration between the total number of tasks realized within a selected processing period.

Solution:

Step 1. Define a new KPI and its parameters according to Table B.8,

Step 2. Output definition:

```
x=Filter[x1<=7];
even=Sum[If[Mod[x,2]==0,1,0]];
odd=Sum[If[Mod[x,2]==1,1,0]];
y=even/odd;
```

Table B.9 FPY KPI parameters

Parameter	Value
Definition name	PassedFailed
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input Datapoints	input1Status as variable x1 input2Status as variable x2

input1Status (B.7)

Description: A Boolean type Datapoint, which corresponds to the state of KIS.Device digital input 1. A similar Datapoint is defined for the second digital input and is called input2Status.

Illustrative example: Let us imagine a quality control process. If a product passes the quality control test, then KIS.Device digital input 1 is fed with the sequence false–true–false. Contrarily, if the product fails this test, then KIS.Device digital input 1 is fed with such a sequence. The objective is to calculate FPY (cf. (4.3)) related to the quality control process within a selected processing period

Solution:

Step 1. Define the new parameters according to Table B.9:

Step 2. Output KPI definition:

```
NumberPassed = RisingEdge[x1];
NumberFailed = RisingEdge[x2];
Total = NumberPassed + NumberFailed;
FPY = If[Total>0, Round[NumberPassed/Total*100], 0];
```

output1Status (B.8)

Description: A Boolean type Datapoint, which corresponds to the state of KIS.Device digital output 1. A similar Datapoint is defined for the second digital output and is called output2Status.

Table B.10 Total mounting task number KPI parameters

Parameter	Value
Definition name	SumComponents
Published as	KPI
Processing period	Select a desired one
Processing start	Select a desired one
Starting hour	Select a desired one
Input Datapoints	output1Status as variable x_1 output2Status as variable x_2

Illustrative example: A worker performs two tasks, i.e., the controller and frame mounting ones. If the first one is completed, then KIS.Device digital output 1 is fed with the false–true–false sequence. Similarly, if the second task is completed, then KIS.Device digital output 2 is fed with the same sequence. The objective is to calculate the total number of realized tasks within a selected processing period.

KPI Solution:

Step 1. Define new KPI parameters according to Table B.10:

Step 2. Output KPI definition:

```
Frames = FallingEdge[x1];
Controllers = FallingEdge[x2];
y=Frames+Controllers;
```

Appendix C

Glossary

System	A part of the universe, that can be affected and/or monitored by KIS.ME.
KIS.MANAGER	The KIS.ME web platform used to affect and/or monitor the system.
KIS.BOX	A communication twin push-button box.
KIS.LIGHT	A communication signal lamp.
KIS.Device	Some KIS.ME hardware, e.g., KIS.BOX, KIS.LIGHT, etc.
KIS.API	A particular instance of REST API, which is designed for the purpose of a unified and convenient communication with KIS.Devices and the associated environment operating within KIS.MANAGER.
Asset	A physical part of the system, which is exemplified by KIS.Devices.
Asset group	A set of assets.
Floorplan	A graphical representation of the workshop.
Datapoint	A read only variable, which corresponds to a possibly time-varying property of a KIS.Device. It can be also defined as an exchanged value between the KIS.Device and KIS.MANAGER.
Dashboard	An overview page for an asset and/or asset groups.
Digital twin	A KIS.MANAGER-based virtual counterpart of a KIS.DEVICE, which is connected to the real one through a dedicated WiFi.
Onboarding	The process of linking KIS.Devices with KIS.MANAGER.
Process	Everything what is needed for transforming an input into an output for a customer.
Processing period	A predefined period of time, which can be either 15, 30 or 60 min.

Calculated Datapoints	FLEX language-based scripts enabling instant processing of Datapoints.
Key performance indicators	FLEX language-based scripts enabling the processing of Datapoints over a predefined processing period.
Rule engine	A KIS.MANAGER functionality which makes it possible to implement functional IF-THEN rules governing interactions between assets.
State	A set of variables which can be used to describe any past and future system behaviour.
User	A human entity with granted access to KIS.MANAGER.
User group	A set of users with a predefined KIS.MANAGER rights level.
Websocket	A communication protocol, which permits bidirectional communication between the client and the web server.
Widget	An interface component which makes it possible to perform a desired action.
Workspace	A selected part of the system which inherits its desired set of assets (asset group).

Index

A

Access control, 79
 external hardware/software, 82
 general, 79
 identification command, 82
 individual, 79
 reset command, 82
Adding a new user, 28
Adding asset groups, 26
Adding dashboard, 39
Admin, 26
Admin and workspaces, 30
Admin user, 24, 72
Aggregation, 118
Agriculture and environmental applications,
 6
Assembly line stations, 96
Asset, 277
 assigning to a group, 34
 binary state, 41
 certificate, 39
 changing name, 34
 device information, 37
 digital input, 41
 digital output, 41
 firmware update, 39
 group relationship graph, 37
 hardware, 39
 info, 34
 management, 33
 network, 39
 software, 39
 time drive, 41, 42, 52
Asset group, 25, 277
 adding, 26

 time drive, 52

Assigning user rights, 28
Assigning users to a group, 32
Associativity of conjunction, 62
Associativity of disjunction, 62
Automated guided vehicle, 194

B

BaseUrl, 213
Bit, 264
Button1ColorKPI, 270
Button1ColorKPIDuration, 269
Button1Pressed, 272
Button2ColorKPIDuration, 269
Button2Pressed, 272

C

Calculate Datapoints (CDP), 278
 sharing, 131
Changing WLAN, 25
Chart
 \bar{x} , 150
 Datapoint, 45
Commutativity of conjunction, 62
Commutativity of disjunction, 62
Container
 capacity, 99
Contraposition law, 62
Count, 252
Counter, 265

D

Dashboard, 277
 Datapoint, 11, 24, 44, 45, 277
 Datapoint chart, 45
 analyzing data, 48
 multiple plots, 45
 storing data, 48
 Datapoint chart widget, 39
 Data sheet widget, 39
 Decision table, 61
 Defect per unit
 significance, 173
 Demerit
 number, 171
 Demerit system, 171
 De Morgan's law, 62
 Digital twin, 277
 design, 41
 Digital twin widget, 39, 41
 Discrete event system, 235
 Distribution
 location, 132
 shape, 132
 variability, 132
 Distributivity of conjunction, 62
 Distributivity of disjunction, 62
 Double negation, 62
 Duration, 261

E

End, 260

F

FallingEdge, 253
 Fault, 205
 Fault-tolerance, 201, 205
 Filter, 267
 First Pass Yield (FPY), 127
 FLEX
 language, 117
 Floorplan, 23, 277
 SVG, 50
 Floorplan widget, 49
 headline, 52
 Full duplex, 235

G

General Purpose Input Output (GPIO), 23
 Group relationship graph, 37

H

Histogram, 146
 bin calculation, 147
 grouping, 147
 Hospitality and leisure industry, 7
 Human–Machine Interface (HMI), 21

I

Idempotency of conjunction, 62
 Idempotency of disjunction, 62
 If, 266
 IF-THEN rules, 23
 Industrial applications, 4
 Info widget, 39
 Initial state, 28
 Input1Status, 275
 Input2Status, 275
 Installer, 26
 Interval, 262
 Inventory asset group, 26
 isOnline, 272

J

Just-in-time strategy, 96

K

Kanban, 98
 number, 98
 Key Performance Indicator (KPI), 11, 24, 41,
 50, 125, 278
 sharing, 131
 KIS.API, 211, 277
 actions, 212
 asset group requests, 215
 asset requests, 215
 calculated Datapoints, 223
 credentials, 213

- Datapoints, 220
- enabledAPI, 229
- Excel, 229
- identifier, 212
- KPIs, 223
- Matlab, 231
- path variables, 214
- registration and authorization, 212
- representation, 211
- resources, 211
- rules, 227
- rule trigger, 228
- status code, 212
- subscription, 237
- user requests, 219
- KIS.BOX, 277
- KIS.Device, 277
- KIS.LIGHT, 277
- KIS.MANAGER, 277
- KIS.ME demo, 25
- KPI aggregated chart widget, 41, 135
- KPI pie chart widget, 41, 136
- KPI single period chart widget, 41, 136
- KPI single value column widget, 41, 135
- KPI single value widget, 41

L

- Led1ColorKPIDuration, 273
- LedColorKPI, 274
- Linguistic objects, 54
- Long polling, 235
- Lower Control Limit (LCL), 149

M

- M12 8-pin, 23
- Material shortages, 98
- MATLAB
 - rule triggering, 233
- Matlab
 - Datapoints, 231
 - webread, 232
- Max, 254
- Mean, 255
 - assembly durations, 135
- Median, 132, 133
- Message Queuing Telemetry Transport (MQTT), 24

- Messaging, 237
- Milk run logistics, 95
- Min, 256
- Mod, 263
- MS Excel
 - power query, 229
- My devices, 26

N

- Negation, 61
- Notification templates, 55
 - variables, 55

O

- Observer, 26
- Onboarding, 24, 277
 - onboarding.zip, 24
 - procedure, 24
- Operator, 26
- Output1Status, 275
- Output2Status, 275
- Overall Equipment Effectiveness (OEE), 11, 177

P

- Parts
 - containers, 97
- Percentile, 133, 257
- Process, 132, 277
 - in control, 133
 - out of control, 133
- Processing period, 117, 224, 277
- Pusher, 242

R

- Range, 133
- Rights and permissions, 28
- RisingEdge, 258
- Rule
 - action, 54
 - antecedent, 54

- automatic simplification, 64
- consequents, 54
- inconsistency, 65
- reduction principle, 66
- redundancy, 65
- Rule base
 - completeness, 67
- Rule-based system, 54
- Rule engine, 12, 23, 54, 278
 - actions, 55
 - conditions, 55
 - device action, 55
 - inference, 55
 - initial condition, 57
 - rule interactions, 58
 - triggers, 55

S

- Safety stocks, 98
- Secure authentication, 24
- Security, 24
- Simple Text-Oriented Messaging Protocol (STOMP), 237
 - heart-beating, 238
- Standard deviation, 133
 - assembly durations, 135
- State, 278
- State-space model, 58
- Statistical control state, 145
- Status LED, 21
- Sstdev, 258
- Streaming, 235
- Sum, 259
- Supermarket, 96
- Supermarket points, 96
- SVG, 50
- System, 277
 - rule-based, 54
 - state, 58
 - state-space, 58
 - state-space model, 58

T

- Tautology, 61

- Traffic lights, 59
- Transfer constraint, 205
- Trigger
 - logistic objects, 55
 - optional settings, 55
- Truth tables, 62

U

- Unique Resource Name, 19
- Upper Control Limit (UCL), 149
- User, 278
- User group, 24, 25, 278
 - membership, 26
- User group permissions, 32

V

- Variance, 133
- Variation
 - common cause, 133
 - special cause, 133

W

- Warehouse management system, 89
- Websocket, 234, 278
- Widget, 39, 278
 - floorplan, 49
- WiFi
 - parameters, 24
- WiFi fingerprint, 7
- Workspace, 25, 278

Z

- Zone, 105
 - floorplan, 105
 - identification, 107