



Design and development of a dynamic IO-Link based framework for
generic sensors

A Thesis

Submitted to the Faculty of Fachhochschule Aachen and Centro de Ingeniería
y Desarrollo Industrial

BY

Víctor Francisco Chávez Bermúdez

In Partial Fulfillment of the requirements
for the degree of Master of Science in Mechatronics

Santiago de Querétaro, Qro., Mexico, September 2018

Declaration

I hereby declare that this thesis work has been conducted entirely on my own accord. The data, the software employed and the information used have all been utilized in complete agreement to the copyright rules of concerned establishments.

Any reproduction of this report or the data and research results contained in it, either electronically or in publishing, may only be performed with the prior sanction of the University of Applied Sciences (FH Aachen), Centro de Ingeniería y Desarrollo Industrial (CIDESI) and myself, the author.

Víctor Francisco Chávez Bermúdez

Santiago de Queretaro, Qro., Mexico, September 2018

Acknowledgment

I want to thank CONACYT for the financial support provided by their scholarship program which allowed me to complete my studies in Mexico and Germany.

I would also like to express my sincere gratitude to Prof. Dr.-Ing. Jörg Wollert from the FH Aachen for the great opportunity he gave me to collaborate and work with him. Likewise, for the valuable support as my thesis advisor through the duration of the project.

Also, I'm grateful for the support brought by my colleagues in the laboratory of the FH Aachen. In the same manner, I appreciate the feedback offered by Dr. Luciano Nava Balanzar as my thesis advisor in Mexico.

Likewise, a special thanks to Brenda, Arantxa and Zurith, who were always there in spite of being hundreds of kilometers away. Finally, to all my friends that have helped me in one way or another through my whole stay in Germany, they have my deepest appreciation.

Dedication

To my parents, who have always been there for me. This chapter in life wouldn't have been completed without you. You taught me that through hard work, perseverance and determination everything is possible. Your values and beliefs are always in my heart no matter where I am.

List of Abbreviations

API	Application Programming Interface
EEPROM	Electrically Erasable Programmable Read-Only Memory
GUI	Graphical User Interface
IODD	IO-Link Device Description File
ISDU	Indexed service data unit
LED	Light-emitting diode
PCB	Printed circuit board
PLC	Programmable logic controller
XML	Extended Markup Language

Abstract

Since the conception of Industry 4.0, there have been different changes in the automation industry. In this manner, sensors have required extra functionalities as transmission of diagnostics and parametrization information. With the purpose of meeting these requirements a standardized communication protocol known as IO-Link was created. IO-Link enables the exchange of valuable information for diagnostics and parametrization of smart sensors in a plug-and-play manner. The current methods for the development of IO-Link sensors are peculiar by the manufacturer and require third-party tools.

This thesis proposes the design and development of a framework based on the IO-Link protocol for generic sensors. The peculiarity of this framework relies on its simplicity and abstraction layer which allows its integration into any type of application. It consists of a user interface for the generation of sensor description files (IODD) and implementation of the IO-Link protocol with the well-known Arduino framework. In addition, it has a hardware interface that acts as the physical layer for the sensor. Finally, different examples are shown to demonstrate the framework's implementation and results.

Resumen

Con la incorporación de la Industria 4.0 ha habido distintos cambios en la automatización industrial. Los sensores actualmente requieren funcionalidades adicionales como transmisión de información de diagnóstico y parametrización. Con el propósito de cumplir estos requisitos, se creó un protocolo de comunicación estandarizado conocido como IO-Link. IO-Link permite el intercambio de información valiosa para el diagnóstico y la parametrización de sensores inteligentes a través de una interfaz “*Plug-and-Play*”. Los métodos actuales para el desarrollo de sensores IO-Link son específicos por fabricante y requieren herramientas de terceros.

Esta tesis propone el diseño y desarrollo de un marco basado en IO-Link para sensores genéricos. La peculiaridad de este marco se basa en su simplicidad y capa de abstracción que permite su integración en cualquier tipo de aplicación. Consiste de una interfaz de usuario para la generación de archivos de descripción de sensores (IODD) y la implementación del protocolo IO-Link con la conocida plataforma de Arduino. Además, cuenta con una interfaz de hardware que actúa como capa física para el sensor. Finalmente, se muestran diferentes ejemplos para demostrar la implementación y resultados finales.

Contents

Introduction	1
Problem Statement	1
Justification	2
Objective	2
Scope	3
Requirements	3
Functional requirements	3
Non-functional requirements	3
Conceptual framework	3
General overview of IO-Link	3
IO-Link system	4
IO-Link messages	6
IO-Link device	7
I/O Device description (IODD)	8
IO-Link device profile	11
Methodology	13
Firmware design and development	13
Hardware requirements	13
Hardware limitations	16
Firmware development	18
Example application for firmware API	26
Firmware operation	27
IODD GUI generator design and development	28

GUI Design	28
Requirements of the configuration file generator	30
Requirements of the IODD file generator	31
Requirements of save and load feature	32
Development of the GUI.....	32
Hardware design and development	46
Results	49
Joystick with LED control.....	49
LED strip controller	55
Distance sensor.....	62
Conclusions	69
Future work	70
Bibliography	71
Annexure 1 Thesis Proposal Outline	72
Annexure 2 Zulassung zur Master-Abschlussarbeit.....	76
Annexure 3 API for IO-Link firmware.....	79
Annexure 4 GUI user manual.....	86

List of Figures

Figure 1 General overview of an IO-Link system in a fieldbus system [4].	4
Figure 2 Three wire connection system.	5
Figure 3 Communication channels in an IO-Link system [3]	6
Figure 4 Typical example of different M-sequences [4]	7
Figure 5 Structures and services of a Device [3]	8
Figure 6 O5D distance sensor.	9
Figure 7 Snippet of IODD profile header description	9
Figure 8 Snippet of IODD device identity information.	10
Figure 9 Snippet of IODD parameter description	10
Figure 10 Snippet of IODD Process data description	11
Figure 11 Typical object model for Device profiles [6]	12
Figure 12 IO-Link Object transfer example [3]	14
Figure 13 TIOL111 development kit [7]	16
Figure 14 Snippet of the Configuration file for the dynamic IO-Link firmware	19
Figure 15 Flowchart of function handler for process output data	20
Figure 16 Flowchart for loading parameter values.	21
Figure 17 Microcontroller pinout functions	22
Figure 18 Modification to the boards.txt file of Arduino IDE software.	23
Figure 19 Board selection in Arduino IDE.	24
Figure 20 Class diagram of the firmware	25
Figure 21 Code snippet of variable declarations for the firmware	26
Figure 22 Firmware initialization	27
Figure 23 Firmware operation in main loop.	27
Figure 24 Code snippet of user function for each cycle process	27
Figure 25 Requirements diagram for the GUI.	28
Figure 26 Use case diagram for the GUI based on the requirements	29
Figure 27 Requirements for the Creation of the configuration file	30
Figure 28 Requirements for the creation of the IODD file	31
Figure 29 Requirements for Save and load feature task.	32

Figure 30 Main window of GUI.....	34
Figure 31 Qt Designer properties dialog	34
Figure 32 Flowchart for displaying errors when generating a file	36
Figure 33 Examples of error notifications	37
Figure 34 Image not supported error	37
Figure 35 Snippet of configuration file template (left) and final output (right).....	38
Figure 36 Example of Sketch file for API reference	39
Figure 37 GUI Class diagram.....	40
Figure 38 Identification parameters tab for the potentiometer device.....	41
Figure 39 Process data tab for potentiometer device.....	42
Figure 40 Custom Parameters tab for potentiometer device	42
Figure 41 File explorer showing the GUI folder output.....	43
Figure 42 Snippet of IODD file of the potentiometer device	44
Figure 43 Snippet of header file of the potentiometer device	44
Figure 44 Snippet of sketch example for the potentiometer device	45
Figure 45 File structure for final distribution of GUI.....	45
Figure 46 Electronic schematic of the transceiver module.....	47
Figure 47 Top layer PCB preview	48
Figure 48 Bottom layer PCB preview	48
Figure 49 3D preview of PCB design.....	48
Figure 50 Data flow for joystick example	49
Figure 51 Identification Parameters in GUI for example 1	50
Figure 52 Process data Tab input for Joystick device	51
Figure 53 Custom parameters Tab input for joystick device.....	51
Figure 54 Arduino sketch for Joystick device	52
Figure 55 Validation of parameters for joystick device	53
Figure 56 Changing custom parameter for joystick device	53
Figure 57 Snippet of PLC program for joystick device	54
Figure 58 Reading axes information from PLC GUI	55
Figure 59 Data flow of LED strip controller	55
Figure 60 LED strip description in GUI.....	57

Figure 61 Process data tab for LED strip	57
Figure 62 Custom parameters tab for LED strip	58
Figure 63 Arduino sketch for LED strip.....	59
Figure 64 Reading and writing parameters to led strip device	60
Figure 65 PLC Program for the control of the LED strip.....	61
Figure 66 GUI for PLC program of LED strip control.....	61
Figure 67 Data flow for distance sensor device	62
Figure 68 GUI Identification parameters for distance sensor.....	63
Figure 69 GUI Process data tab for distance sensor	63
Figure 70 GUI Event tab for distance sensor	64
Figure 71 Arduino sketch for distance sensor	65
Figure 72 Test of IO-Link events triggered by the distance sensor.....	66
Figure 73 Reading IO-Link parameters of the distance sensor	67
Figure 74 WAGO PLC program for the distance sensor	67
Figure 75 GUI for PLC program of the distance sensor.....	68
Figure 76 Main window of GUI.....	87
Figure 77 Identification parameters tab.....	88
Figure 78 Process data Tab.....	91
Figure 79 Custom parameters input for the user in GUI	92
Figure 80 Events tab	93
Figure 81 Custom events tab	94

List of Tables

Table 1 Microcontroller development kits comparison for IO-Link firmware	15
Table 2 IO-Link transceivers comparison	16
Table 3 Rough estimations of compiled firmware size	24
Table 4 Use case description of GUI.....	30
Table 5 IO-Link potentiometer description	41
Table 6 BOM of electronic schematic	47
Table 7 Joystick with LED control device description.....	50
Table 8 LED strip controller device description	56
Table 9 Distance sensor device description.....	62
Table 10 General functions of GUI	88
Table 11 Parameter identification Tab Options.....	90
Table 12 Process data Tab options	92
Table 13 Custom parameters tab options	93
Table 14 Events tab options	94
Table 15 Custom events tab options.....	94

Introduction

Over the years, the advancements in manufacturing and information technology have brought up new concepts into the industrial sector. One of these novel concepts is the well-known Industry 4.0 revolution. It was originated in Germany with the purpose of implementing new technology strategies as part of their economic policies in 2011 [1].

Industry 4.0 is composed of the integration of different technologies, such as cloud systems, embedded systems, data analytics, adaptive robotics, cybersecurity and sensors [2]. In that matter, it makes sense to have as much information as possible to handle it and involve a full diagnosable and deterministic system.

To this extent, sensors, which normally just carry a binary or analog signal require more flexibility with the data they transmit e.g. diagnostic information, parametrization, and complex data structures. This necessity resulted in the creation of the standardized communication interface known as IO-Link under the name of Single-drop digital communication interface for small sensors and actuators (SDCI) as part of the norm IEC 61131-9.

IO-Link incorporates the data flexibility required for Industry 4.0 with the transmission of diagnostics data, parametrization of the device and organized data structures. Hence it offers sensors troubleshooting capabilities, minimal risk of failure and remote diagnostics, and leads them to the concept of “smart sensors”.

Problem Statement

At the FH Aachen, there exists a great interest on the development of automation projects. There is a laboratory which develops projects in the frame of Industry 4.0. As a result, their projects could benefit from the use of the IO-Link communication system for improvements in performance and technology. Unfortunately, there aren't tools that provide a low-cost and open-source framework for the development of prototyping IO-Link solutions. The currently available options are third-party, which have license restrictions and do not offer a complete integrated solution. Hence, the workflow for the implementation of the IO-Link protocol is not straightforward and does not offer

one unique methodology for the development of prototypes and tests on the fly. Furthermore, an IO-Link sensor requires the use of a so-called IODD file. This file is unique per sensor, so a tool that can intuitively create these files is required.

Justification

This thesis proposes a solution to facilitate the implementation of the IO-Link communication protocol for generic sensors through the design of a dynamic framework. This will help decrease the time of design and implementation of generic sensors with the IO-Link protocol.

In addition, this approach also creates an abstraction layer for users to ease up the utilization of the IO-Link communication protocol and focus more on the design and development of their applications.

Furthermore, a framework capable of being customizable implies that virtually any device application can be implemented within the IO-Link protocol. Consequently, it facilitates the integration of more devices to the latest automation industry standards.

Objective

To design and develop an IO-Link based framework composed of a dynamic firmware and user interface for generation of IODD files.

Scope

The aim of this work is to design and develop an IO-Link framework that can serve as a low-cost developing environment for prototype applications. Consequently, this project will focus on the design and development of the firmware that manages the IO-Link protocol, the hardware platform that will embed the firmware and the software for the generation of IODD files.

Requirements

Functional requirements

- Custom firmware for the IO-Link communication protocol.
- Low-cost hardware solution for the embedded IO-Link firmware.
- Software utility for the generation of IODD files.

Non-functional requirements

- IO-Link communication firmware must support the core functionalities of the IO-Link specification described in [3].
- Capability of firmware to change core parameters of the IO-Link specification (e.g. data size, default values).
- IODD file generator shall be fully customizable for the end-user.
- IODD file generator must be developed as a stand-alone software.

Conceptual framework

General overview of IO-Link

The IO-Link protocol was first presented at the Hannover Industrial fair in 2006. The aim was to develop a universal communication interface that transmitted analog and digital signals, and to exchange device parameters with overlying controllers in a system [4].

The first complete specification for IO-Link products was released in 2008 as version 1.0. Figure 1 shows an overview of how the implementation of an IO-Link system looks, where the information is sent to a gateway (e.g. PLC) which then is distributed through a fieldbus interface.

An IO-Link system consists of a “master” which retrieves all the available information from the devices connected to it (e.g. sensors). In turn, this information can be linked to a gateway (e.g. PLC) connected to a fieldbus interface.

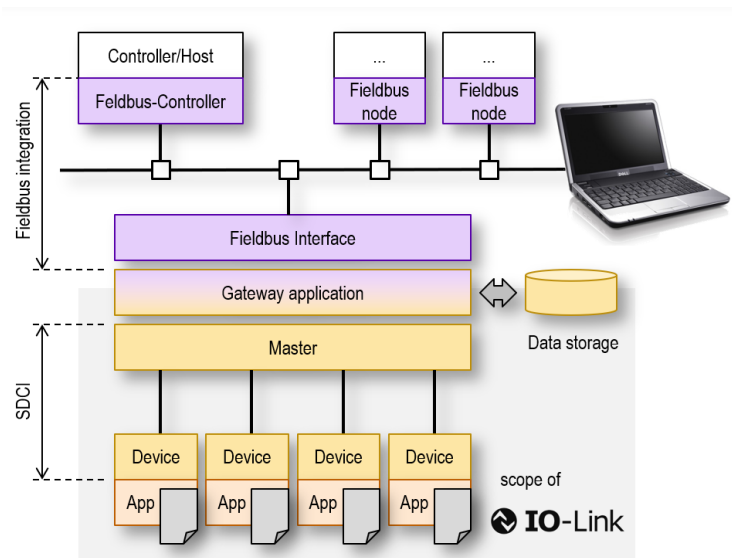


Figure 1 General overview of an IO-Link system in a fieldbus system [4].

IO-Link system

“IO-Link is the first I/O technology for communicating with sensors and actuators to be adopted as an international standard (IEC 61131-9).” [5]. It is a point-to-point communication that consists of the following elements.

- IO-Link master
- IO-Link device

For each IO-Link system, there exists one IO-Link master which can connect with an automation system and with one or more IO-Link devices. The physical connection between device and master

is done through a 3-wire connection as seen in Figure 2. The L+ wire is used as a 24 V power supply, L- as ground and C/Q for the communication signal.

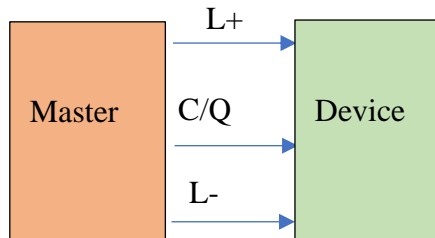


Figure 2 Three wire connection system

Two different types of data are sent between an IO-Link master and device: process data (sent cyclically) or on-request data (sent acyclically). This data is structured into three communication channels:

- **Operational data**

This data is cyclically transmitted from the device to the master (process input data) and vice versa (process output data), depending on the device-specific technology, the size may vary but cannot be greater than 32 bytes.

- **Configuration/maintenance**

This data is transmitted acyclically and serves as a mean to read and write parameters from and to the device.

- **Events**

This channel transmits diagnostic information from the device in an acyclic manner

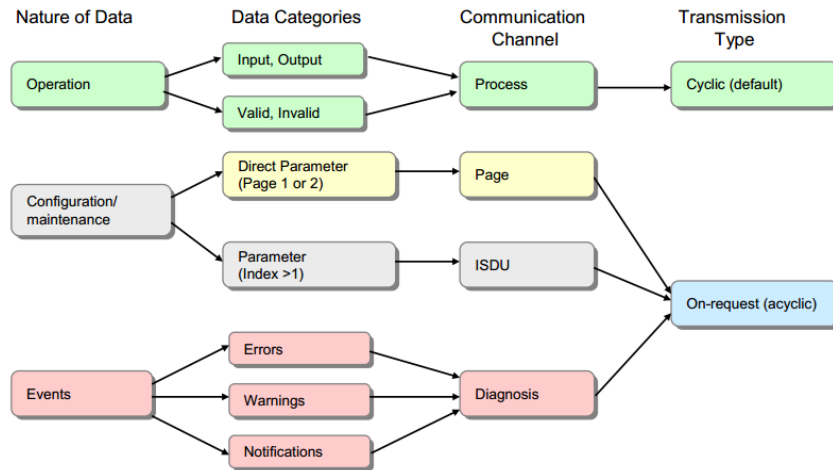


Figure 3 Communication channels in an IO-Link system [3]

IO-Link messages

The IO-Link communication is always started from the master and ends with the response of a device. The sequence of two messages between a master and device is called an M-sequence. These sequences are sent in a cyclical manner, where the minimum cyclic time supported depends on the device.

The first byte that an IO-Link master sends in a UART frame is called an M-sequence control code (MC). This data indicates to the device if a read or write operation is going to be performed, the communication channel to be used and the address where the information is sent. Then if the IO-Link device is set to “operation” mode the master will send process output data (if supported by the device), and finally, if a write operation is indicated from the MC it will send acyclic data (on-request data).

The response of the device will start with on-request data if the MC requested a read operation, if not it will send process data (if “operation” mode is set) and finally a checksum byte to validate the data. An example of how a whole m-sequence may look like is shown in Figure 4.

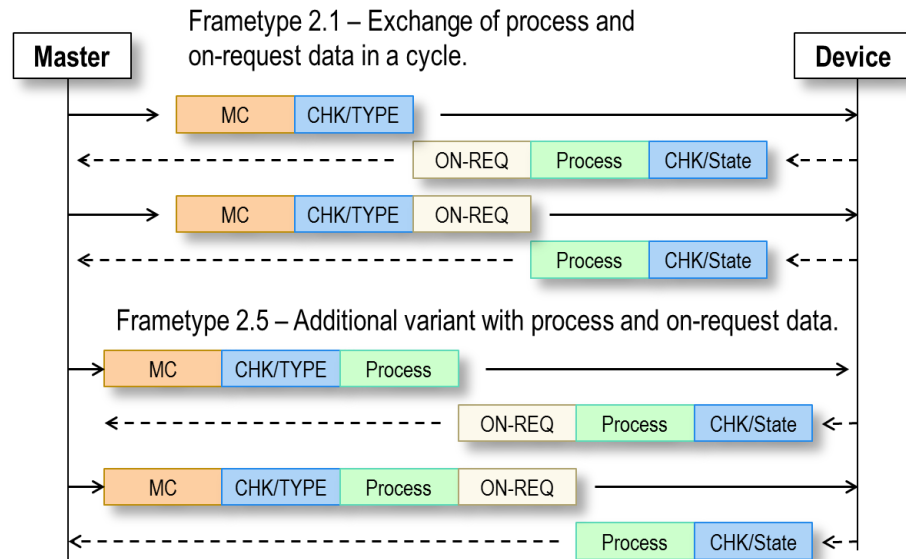


Figure 4 Typical example of different M-sequences [4]

IO-Link device

The IO-Link protocol for a device is structured in layers that are in charge of specific tasks for the correct operation of the system. These layers are structured as follows:

- **Application Layer:** Supervises and controls the read and write operations of the device.
- **Data Layer:** Redirects the current received data to the corresponding communication channel.
- **Physical Layer:** Interaction between the device's software layer and the external physical layer.

These layers (see Figure 5) communicate internally to process data and command operations. The operations of the device are handled by the so-called device applications. The device applications are divided as follows:

- **Parameter Manager:** Checks the validity and range of parameters that are written to the device.

- **Data storage:** If enabled, it stores the specified parameters in the non-volatile memory of the device.
- **Event Dispatcher:** Used to signal events that are originated by the device.
- **Process data Exchange:** In charge of the transmission and reception of process data from the master to device and vice-versa.

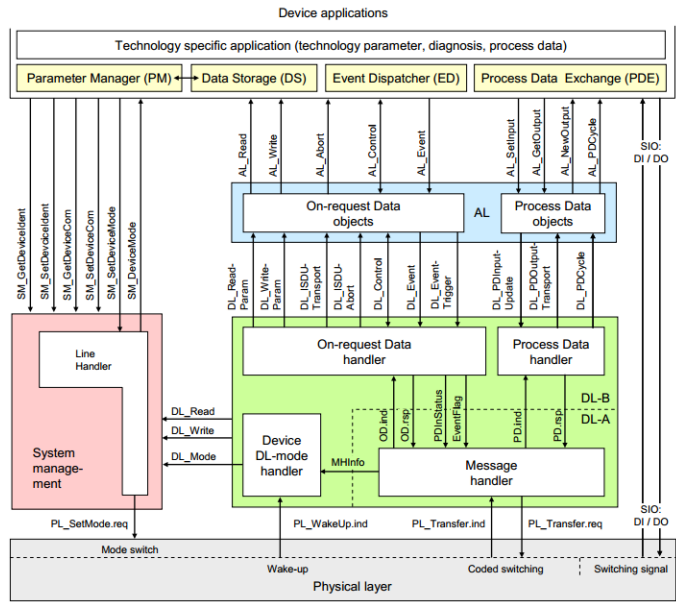


Figure 5 Structures and services of a Device [3]

I/O Device description (IODD)

The IODD is a mandatory XML file encoded in “UTF-8”, that establishes all the required parameters for the communication between a device and a master. It also defines the parameters that can be changed by a user and modified via an IODD tool, which is usually contained in the software utility of an IO-Link master.

As an example, the IODD file from the distance sensor O5D from IFM electronics is shown (see Figure 6). Figure 7 shows the first section of the IODD file, which contains the general description of the file, which includes the file version of the vendor, and the IO-Link protocol version.

The next section of this IODD file contains the specific device information from the sensor, e.g. vendor name, device name (see Figure 8). Consequently, a device may be parameterized, Figure 9 shows a snippet as how the data type, size, and name are described. Finally, the process data

structure is shown in Figure 10, this part is structured in length and type of each process variable and includes a descriptive name.



Figure 6 O5D distance sensor

```
<DocumentInfo version="V1.0.3" releaseDate="2015-08-04" copyright="Copyright 2015, ifm electronic gmbh"/>
<ProfileHeader>
  <ProfileIdentification>IO Device Profile</ProfileIdentification>
  <ProfileRevision>1.1</ProfileRevision>
  <ProfileName>Device Profile for IO Devices</ProfileName>
  <ProfileSource>IO-Link Consortium</ProfileSource>
  <ProfileClassID>Device</ProfileClassID>
  <ISO15745Reference>
    <ISO15745Part>1</ISO15745Part>
    <ISO15745Edition>1</ISO15745Edition>
    <ProfileTechnology>IODD</ProfileTechnology>
  </ISO15745Reference>
</ProfileHeader>
<ProfileBody>
```

Figure 7 Snippet of IODD profile header description

```
<DeviceIdentity vendorId="310" vendorName="ifm electronic gmbh" deviceId="393">
  <VendorText textId="TI_VendorText"/>
  <VendorUrl textId="TI_VendorUrl"/>
  <VendorLogo name="ifm-logo.png"/>
  <DeviceName textId="TI_DeviceName"/>
  <DeviceFamily textId="TI_DeviceFamily"/>
  <DeviceVariantCollection>
    <DeviceVariant productId="05D101" deviceSymbol="ifm-000189-05D10x-pic.p
      <Name textId="TN_Product1"/>
      <Description textId="TD_Product1"/>
    </DeviceVariant>
    <DeviceVariant productId="05D151" deviceSymbol="ifm-000189-05D15x-pic.p
      <Name textId="TN_Product2"/>
      <Description textId="TD_Product2"/>
    </DeviceVariant>
  </DeviceVariantCollection>
</DeviceIdentity>
```

Figure 8 Snippet of IODD device identity information

```
<Variable id="V_BDC1_SP" index="60" accessRights="rw" dynamic="true">
  <Datatype xsi:type="RecordT" bitLength="32" subindexAccessSupported="t
    <RecordItem subindex="1" bitOffset="16">
      <SimpleDatatype xsi:type="UIntegerT" bitLength="16">
        <ValueRange lowerValue="2" upperValue="80"/>
      </SimpleDatatype>
      <Name textId="TN_BDC1_SP1"/>
    </RecordItem>
    <RecordItem subindex="2" bitOffset="0">
      <SimpleDatatype xsi:type="UIntegerT" bitLength="16">
        <SingleValue value="0">
          <Name textId="TN_BDC1_SP2_0"/>
        </SingleValue>
      </SimpleDatatype>
      <Name textId="TN_BDC1_SP2"/>
    </RecordItem>
  </Datatype>
  <RecordItemInfo subindex="1" defaultValue="40" />
  <RecordItemInfo subindex="2" defaultValue="0" excludedFromDataStorage='
  <Name textId="TN_BDC1"/>
  <Description textId="TD_BDC1_SP"/>
</Variable>
```

Figure 9 Snippet of IODD parameter description

```
<ProcessDataCollection>
  <ProcessData id="V_PdIn">
    <ProcessDataIn id="V_PdInT" bitLength="16">
      <Datatype xsi:type="RecordT" bitLength="16" subindexAccessSupport
      <RecordItem bitOffset="4" subindex="1">
        <SimpleDatatype xsi:type="UIntegerT" bitLength="12">
          <ValueRange lowerValue="2" upperValue="80"/>
        </SimpleDatatype>
        <Name textId="TN_PDV1"/>
        <Description textId="TD_PDV1"/>
      </RecordItem>
      <RecordItem bitOffset="0" subindex="2">
        <SimpleDatatype xsi:type="BooleanT">
          <SingleValue value="false">
            <Name textId="TN_PDV2_false"/>
          </SingleValue>
          <SingleValue value="true">
            <Name textId="TN_PDV2_true"/>
          </SingleValue>
        </SimpleDatatype>
        <Name textId="TN_PDV2"/>
        <Description textId="TD_PDV2"/>
      </RecordItem>
    </Datatype>
    <Name textId="TN_PD_In"/>
  </ProcessDataIn>
</ProcessData>
</ProcessDataCollection>
```

Figure 10 Snippet of IODD Process data description

IO-Link device profile

Sensors in automation systems send data related to measurements done in the environment they interact. Nowadays with the inclusion of Industry 4.0 to these systems, this information is not enough. This has required adding functions for identification, diagnosis, and parametrization. With the purpose of categorizing these features, the IO-Link consortium encloses these common functions into profiles.

These profiles establish functions, data structures, and parameters that an IO-Link device may have. Figure 11 shows an example of how the structure of an IO-Link device with a device profile may look.

Each device can have a profile characteristic, which contains up to 32 entries [6]. These entries consist of “Function classes” (i.e. functionalities a device may support) and/or device profile identifiers which can contain a set of function classes.

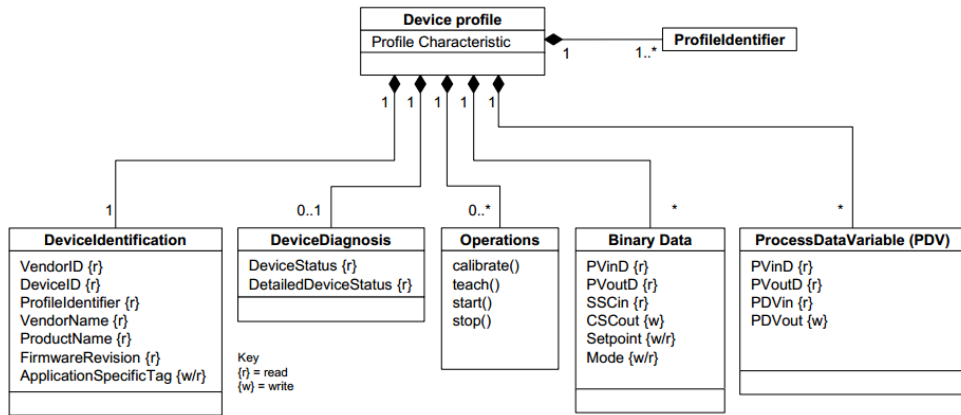


Figure 11 Typical object model for Device profiles [6]

Methodology

The development of the project was done in successive steps that were integrated together. In the first section the design and development of the IO-Link communication firmware are presented. The firmware's main characteristics and limitations were established, and the general structure of how it functions is displayed. Furthermore, a short example that explains the workflow of the firmware's operation is shown.

Then the design and development of the IODD file generator are discussed. This section shows its design requirements, external dependencies that were needed and a practical example on how it can be used to create IODD files for generic sensors.

Afterward, the design and development of the hardware for the framework is presented. It describes the design of the hardware layer for the communication protocol and development of a prototype PCB.

In addition, the implementation of the framework is presented with the purpose of creating a proof of concept. This section shows different examples to create an IODD file, a program for testing the IO-Link device with the firmware API, and its interaction with an IO-Link master. Finally, in the last section a review of the work and conclusions with an outlook for future improvements is discussed.

Firmware design and development

Hardware requirements

The firmware is contained in an embedded processor that is the backbone of the whole system. For this reason, it is necessary to analyze the hardware requirements for it.

Firstly, an IO-Link device can have either parameters for its replacement or extra functionalities, which the user can change through an IODD tool. These parameters can have either read-only

access or read-write access. In other words, the system needs to have volatile and non-volatile memory support.

Secondly, the IO-Link protocol is a point-to-point communication based on a universal asynchronous receiver-transmitter (UART). According to the specifications on [3] the hardware shall have at least a half-duplex UART with support for either 4.8 kbit/s, 34.8 kbit/s or 230.4 kbit/s.

Furthermore, the IO-Link communication protocol consists of an object-like structure with internal and external services (see Figure 12). Hence, it is appropriate that the system can support an object-oriented programming (OOP) approach.

As a short summary, the framework shall meet the following hardware requirements:

- Volatile/Non-volatile memory support
- Half-duplex UART (4.8 kbit/s, 34.8 kbit/s or 230.4 kbit/s)
- OOP support

Analyzing the aforementioned requirements, a microcontroller results in the best option since nowadays they include all these features. Since the main purpose of the project is to have a low-cost open-source tool for the development of prototypes, different microcontrollers were compared to select the most suitable (Table 1).

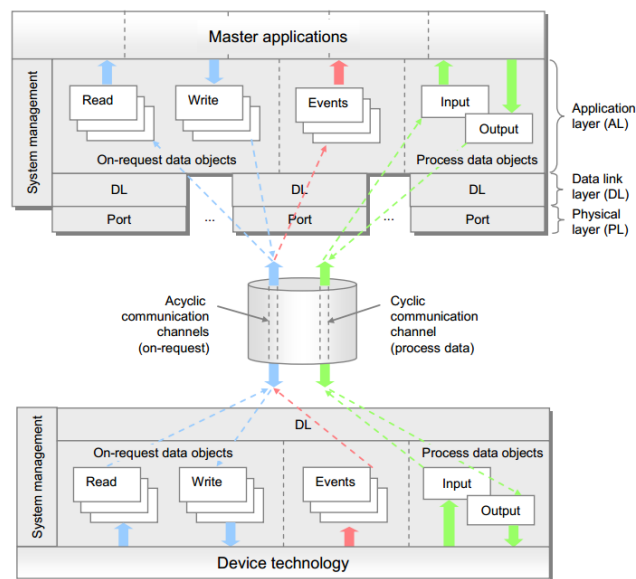


Figure 12 IO-Link Object transfer example [3]

Manufacturer	Atmel (Arduino Nano)	Texas Instruments (Launchpad MSP430)	STMicroelectronics (STM32 Discovery)
Microcontroller Line	Atmega328p (8-Bit)	MSP430FR5994 (16-Bit)	STM32F103x4 (32-Bit)
Program Memory size (KB)	32	264	32
SRAM (KB)	2	8	10
Non-volatile Memory (KB)	1024	256	Emulated (variable)
Max. Clock freq. (MHz)	16	16	72
Cost (USD)	\$4.00	\$17.64	\$11.36

Table 1 Microcontroller development kits comparison for IO-Link firmware

As seen from the previous comparison table, the biggest difference between the distinct products is, in fact, the price. The Texas Instruments and STMicroelectronics option offer a good amount of resources but their market price is roughly the triple and double respectively, in comparison to the Atmel alternative.

Hence, the Arduino Nano development kit was selected for the implementation of the framework. It has the appropriate resources for the firmware implementation and it is low-cost.

The next step is selecting an IO-Link transceiver to interface the UART from the microcontroller to the IO-Link physical layer. There are several options in the market that provide transceivers; the characteristics of three different vendors are described in Table 2.

Vendor	Maxim Integrated	STMicroelectronics	Texas Instruments
Transceiver	MAX14827	L6362A	TIOL111-5
Control Interface	SPI (serial programming interface)/ Digital Pin	Digital Pin	Digital Pin
Pins	24	12	10
Size	4 x 4 mm	3mm x 3mm	2.5 x 3.0 mm

Output voltage	5V and 3.3V @50mA to 250mA	5V or 3.3V @10mA	5V @50mA to 350mA
Protection functions	<ul style="list-style-type: none"> • Reverse polarity • Thermal protection 	<ul style="list-style-type: none"> • Reverse polarity • Overload with cut-off function. • Thermal protection • Surge protection • GND and VCC open wire 	<ul style="list-style-type: none"> • Reverse Polarity • EMC Protection • Surge protection • Thermal Protection

Table 2 IO-Link transceivers comparison

The selection of the transceiver was based upon the chip that could offer a minimal design setup and at least 200 mA of output current to power up the microcontroller and its sensor application.

The MAX14827 falls short on simplicity and the L6362A does not provide enough output current. Therefore, the TIOL11-5 (see Figure 13) was the best option for this case since it has the lowest number of pins, enough output current and in addition integrated protection.

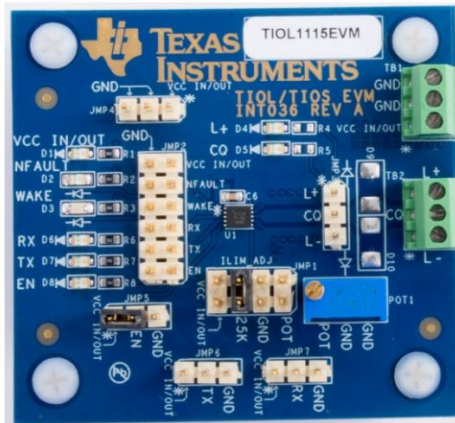


Figure 13 TIOL111 development kit [7]

Hardware limitations

Because the design of the firmware is dependent on the characteristics of the embedded processor, it is necessary to know the hardware limitations of the microcontroller to delimit the capabilities of the firmware.

In general, memory allocations and data assignment are among the main challenges when working with embedded systems [8]. To simplify this problem all the objects and variables were allocated in memory at compile time (static memory allocation).

As mentioned before, the device can include a data storage mechanism, which parametrizes the device for any eventual replacement in case of failure. The atmega328p includes an electrically erasable programmable read-only memory (EEPROM) that can serve this purpose.

This EEPROM takes 3.3 milliseconds for writing a byte into it. As a result, the number of parameters a user may add to the data storage mechanism shall be limited since the time it takes for writing operation is shared with the specific device application (sensor sampling, filters, etc.) and can block the user's main program.

Finally, the maximum baud rate was calculated in order to know which COM mode (described in [3]) would be used. The UART speed of the atmega328p is configured by initializing two 8-bit registers which are part of a 16-bit value. This value can be calculated depending on the desired UART speed with the following formula:

$$UBBR0 = \frac{f_{osc}}{16 * BAUD} - 1$$

Where

UBBR0 : 16 bit value to calculate

f_{osc}: Atmega328p oscillator frequency (16 MHz for the Arduino framework)

BAUD: Desired baud rate

According to table 24-2 in [9], it is recommended that the maximum baud rate error for 8 bits of data and 1 parity bit (IO-Link specification) should be $\pm 1.5\%$.

Calculating *UBBR0* for a baud rate of 230.4 kbit/s (COM3) gives a value of 3.34. Since the microcontroller only accepts integer values the calculated value is rounded to 3. Calculating the physical baud rate with this value is 250 kbit/s, which gives a baud rate error of 8.5 %. This means that COM3 mode is not possible to use.

The next speed is 38.4 kbit/s (COM2), calculating UBBR0 gives a value of 25.04, which is rounded to 25. Calculating the physical baud rate gives a speed rate of 38.462 kbit/s. The baud rate error of this mode is 0.2%, which is inside the range tolerance for correct operation of the device. Therefore, COM2 was used.

Firmware development

As mentioned before, an IO-Link device consists of a series of parameters, used for storing device functionalities and data. One of the main functionalities the firmware must have is a way of accessing these parameters with a specific index and sub-index when the IO-Link master requests them.

Since the parameters can be accessed in any order, each time the IO-Link master requests a write or read action to any parameter the IO-Link device must search in its memory this parameter.

To simplify this process a lookup table with the location of the parameters in the static memory was used. The values of the lookup table are not always sorted since the user can change them, therefore a linear search method is used for finding values in the lookup table [10].

Furthermore, the design of the firmware can be modified accordingly so the final user can change configuration parameters of the IO-Link protocol. Some of these modifiable parameters are:

- Size of process input data
- Size of process output data
- Size and initial value of ISDU Parameters
- Data storage size

These modifiable parameters are specified in preprocessor directives in a header file, which is read at compile time of the firmware (see Figure 14), which allows changing functionalities of the system depending on user requirements. Since the manual modification of a header file is not intuitive and time-consuming, a graphical user interface (GUI) was developed to ease up the process, which is covered in an upcoming section.

```
/*
User Configuration file for IO-Link device options
*/
#ifndef CONFIG_H
#define CONFIG_H
/*-----
User defined parameters for specific application
-----*/
#define GENERIC_SENSOR//Smart Sensor profile Mode
//-----
//Define Data storage if needed
#define MINCYCLETIME 86//Defined in B.1.3 Cycle Time = 15.2ms
//Defined in B.1.6
#define Length_PDOut 0//Min value 1
#define Length_PDIn 1//Min value 1
#define sizeOD_PREOPERATE 8 //Values should be 1,2,8 or 32
#define sizeOD_OPERATE 8//Values should be 1,2,8 or 32
#define specificTechParam 0//Number of specific technology parameters
#define numberSpecificTechParamToSave 0//Number of custom parameters to save in
#define SPECIFIC_TECH_PARAM_SIZE_DS 0//Total size in octets of custom parameter
#define sizeVendorName 2//Number of characters of vendor Name Max 64
#define sizeProductName 15//Number of characters of Product Name Max 64
#define VENDOR_ID 9999//Default Vendor ID
#define DEVICE_ID 1234//Default Device ID
/*-----*/
#if defined(FSS) || defined(AdSS) || defined(DMS) || defined (GENERIC_SENSOR)
#define DATASTORAGE_ENABLED
#define SMART_SENSOR_PROFILE //Needed for FSS,AdSS or DMS
#define SIZE_APP_TAG 3//
#define SIZE_FUN_TAG 3//
#define SIZE_LOC_TAG 3//
#define APP_SPEC_TAG "Pro"//Default App specific tag
#define FUNCTION_TAG "Gen"//Default Function tag
#define LOCATION_TAG "Lab"//Default Location Tag
#endif
#endif
```

Figure 14 Snippet of the Configuration file for the dynamic IO-Link firmware

Additionally, the IO-Link device needs to send and receive process data once it has established communication with the IO-Link master. To update the process input data a method for handling the synchronization of each new message was implemented following the IO-Link interface recommendations.

The IO-Link protocol notifies the application layer each time a new process output data message has been received, this notification is used to update the sensor data through a function handler which calls the user's device specific function for reading and sending the new sampled data from the sensor (see Figure 15).

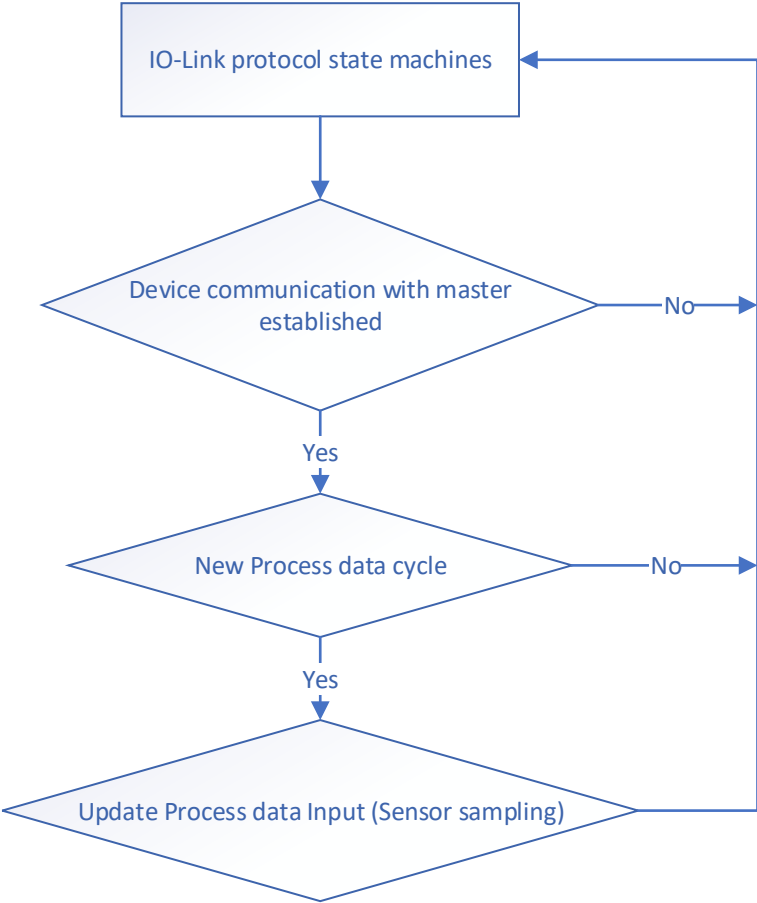


Figure 15 Flowchart of function handler for process output data

Another consideration for the data storage mechanism is the dynamic addition of custom parameters saved on the non-volatile memory. Each new parameter is saved in the preferred index range as described in [3] where each new custom parameter is added accordingly.

In the case the data storage mechanism is enabled, the parameters are initialized for the first time with default values and in any subsequent restart, these values are loaded from the EEPROM. Hence, the user shall indicate in the microcontroller if it's the first time the firmware will be used by writing a value of 1 to a specific address in the EEPROM which acts as a flag for the system (see Figure 16)

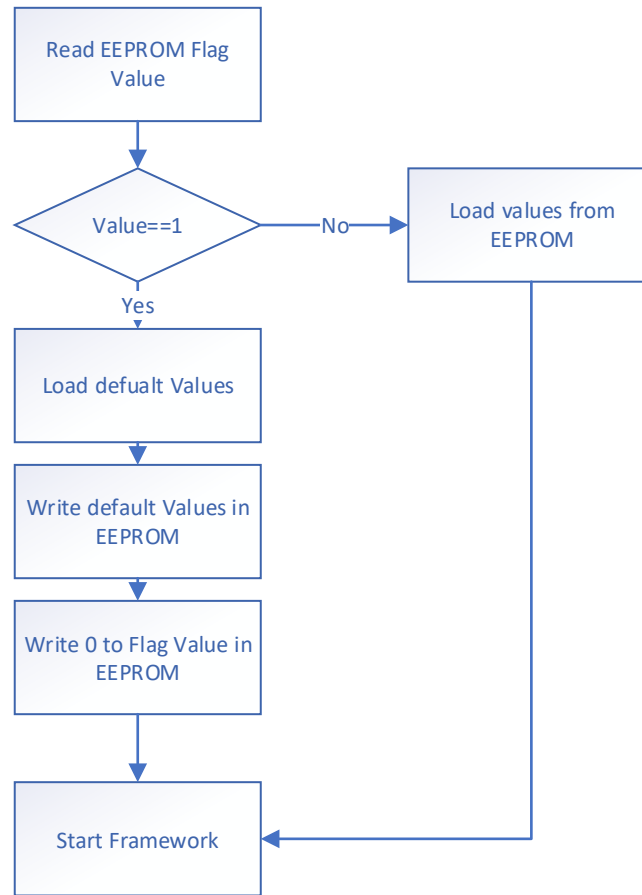


Figure 16 Flowchart for loading parameter values

Finally, to interact with the TIOL111-5 transceiver, it was required to use the external interruption functionality of the atmega328p to detect when the IO-Link master wants to communicate with the device through a wake-up pulse which is described in [3].

In addition, a digital output was used to control the TIOL111-5 transceiver “Enable” pin for either reading or sending data since the IO-Link protocol works as a half-duplex UART. As a summary Figure 17 shows the microcontroller ports that were used for the firmware.

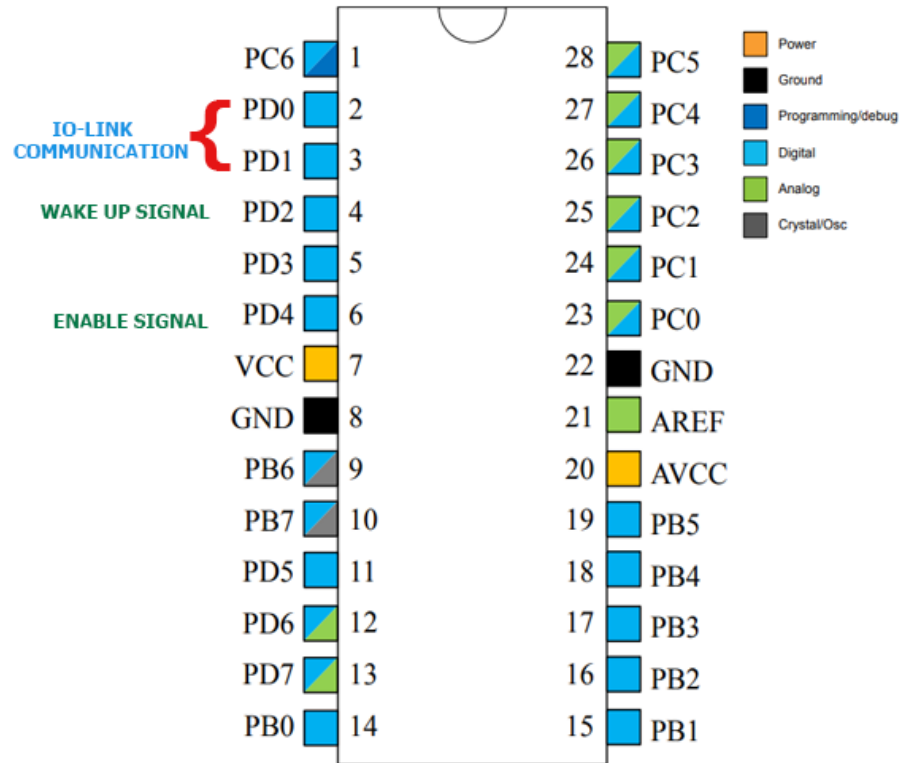


Figure 17 Microcontroller pinout functions

Additionally, as described in [11], it is quite useful to have measurement results from different sensors in a uniform manner. This brought the addition of the so-called “smart sensor profiles” which are device profiles specifically designated for sensors that transmit more type of information.

The use of these profiles can speed up the inclusion of common functionalities for the development of IO-Link. For this reason, it was considered that the firmware should support the use of the next smart sensor profiles:

- Fixed switching sensors (FSS)
- Digital measuring sensors (DMS)
- Generic Sensors

As aforementioned, the IO-Link software library works with a configuration file which the user may change. The easiest approach is that this file should be contained in the folder where the user’s

main program is located. Since it is used with the Arduino framework this configuration file should be inside the folder of the so-called “sketch” file.

The compilation options of the Arduino framework must be modified so it can compile this external header file from the sketch folder of the user. To achieve this, the file “board.txt” inside the Arduino IDE software installation has to be modified (see Figure 18) so it includes the configuration file as part of the compilation of the software library.

Once this modification is made, from the Arduino IDE the user only needs to change the “board” for the new one that has been added through the menu bar Tools\Board\Arduino Nano (IO-Link) (see Figure 19).

```
#####  
ioLink.name=Arduino Nano (IO-Link)  
  
ioLink.upload.tool=avrdude  
ioLink.upload.protocol=arduino  
  
ioLink.bootloader.tool=avrdude  
ioLink.bootloader.unlock_bits=0x3F  
ioLink.bootloader.lock_bits=0x0F  
  
ioLink.build.f_cpu=16000000L  
ioLink.build.board=AVR_NANO  
ioLink.build.core=arduino  
ioLink.build.variant=eightanaloginputs  
ioLink.build.extra_flags=-include "{build.path}/sketch/config.h"  
  
## Arduino Nano w/ ATmega328  
## -----  
ioLink.menu.cpu.atmega328=ATmega328  
  
ioLink.menu.cpu.atmega328.upload.maximum_size=30720  
ioLink.menu.cpu.atmega328.upload.maximum_data_size=2048  
ioLink.menu.cpu.atmega328.upload.speed=57600  
  
ioLink.menu.cpu.atmega328.bootloader.low_fuses=0xFF  
ioLink.menu.cpu.atmega328.bootloader.high_fuses=0xDA  
ioLink.menu.cpu.atmega328.bootloader.extended_fuses=0x05  
ioLink.menu.cpu.atmega328.bootloader.file=atmega/ATmegaBOOT_168_atmega328.hex  
  
ioLink.menu.cpu.atmega328.build.mcu=atmega328p
```

Figure 18 Modification to the boards.txt file of Arduino IDE software

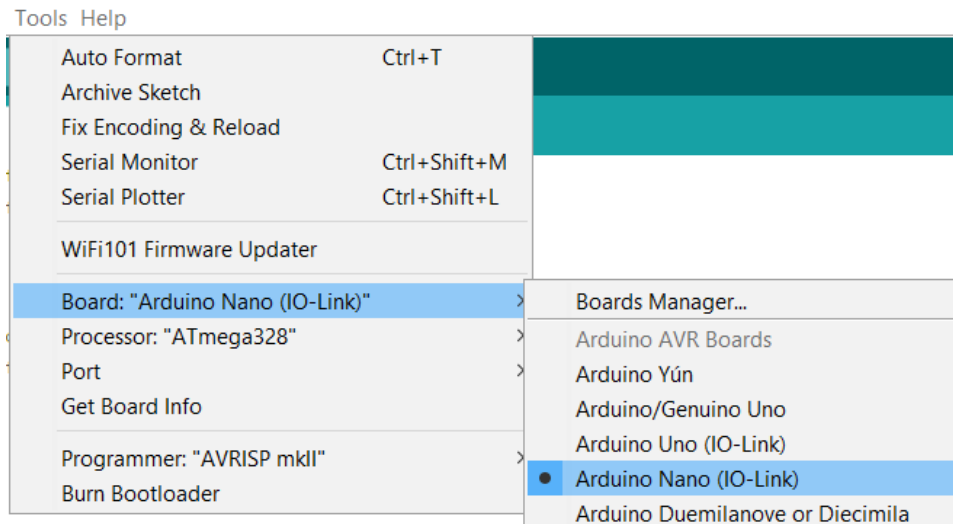


Figure 19 Board selection in Arduino IDE

Finally, the firmware library relies on having an object device in charge of managing all the state machines that are in charge of the communication layers and informing of the relevant flags for the user's specific application. Figure 20, shows a Class Diagram that gives an overview of the relations between the communication and process layers of the IO-Link device for the framework.

As a final remark, the firmware was compiled with two general configurations to estimate roughly its size, the results are shown in Table 3.

Firmware Configuration	Flash Memory (KB)	SRAM (KB)
<ul style="list-style-type: none"> No data storage No custom parameters No device profile 	9.39	0.78
<ul style="list-style-type: none"> Data storage Custom parameters Smart sensor profile 	13.30	0.98

Table 3 Rough estimations of compiled firmware size

Example application for firmware API

This section describes how the API of the firmware (see Annexure 3 API for IO-Link firmware) works with an application example to demonstrate the workflow for developing any application.

Firmware Setup

Firstly, as seen in Figure 21, an object of the type “io_link_device” shall be initialized which contains the available methods from the API. The next step is to declare all the custom parameters that the user might occupy as variables, for this example it is assumed the device has one custom parameter of 8 bits. Afterward, if the application shall send data to the master (i.e. process input data), a global variable that will contain device data must be declared.

```
#include <IO-Link Device.h>

io_link_device arduinoIOLink; //IO-Link object
uint8_t customParameter; //Declaration of user custom parameter
uint8_t deviceData; //Global to send device data to Master
```

Figure 21 Code snippet of variable declarations for the firmware

Secondly, the firmware must be initialized as seen in Figure 22. For this example, the method “addParameter” shall be called once for each custom parameter, this allows the framework to know which variable shall be used for read/write operations performed by the IO-Link master.

After initializing the custom parameters, the method “begin” notifies the framework that it’s the first time the program is running with the purpose of loading or not the default values of custom parameters. Then the method “initDevice” is called, the first parameter that this method accepts indicates the user function that should be called each time a cyclic process is completed by the IO-Link master. In this function, the user shall implement its specific application. The second parameter for this method is optional and indicates whether process input data is available from the device (for this example the variable “deviceData” is used).

```
arduinoIOLink.addParameter(&customParameter,sizeof(customParameter),true);  
  
arduinoIOLink.begin(); //Should be uploaded only once and then removed on next upload  
arduinoIOLink.initDevice(sensorTask,deviceData); //Init IO-Link Device Settings
```

Figure 22 Firmware initialization

The firmware operation consists of executing the firmware's function that takes control over the IO-Link protocol and ensures the proper communication with the IO-Link master (see Figure 23). To simplify this process the method "ioLink_Task" shall be called once for each time in the main loop of the program. Additionally, the user can call the method "isComLost" to check at any time if the IO-Link master is disconnected to perform any specific task.

```
void loop() {  
    arduinoIOLink.ioLink_Task();  
    if(arduinoIOLink.isCOMLost()){  
        //Do something if communication with IO-Link master is lost  
    }  
}
```

Figure 23 Firmware operation in main loop

The other part of the firmware operation consists of the execution of the user function that is called per cycle process (see Figure 24). If the user application has process output data (i.e. data sent to the device) the user may use the method "getPDOOut" to retrieve data that the IO-Link master sends on each process cycle. To ensure the process output data is valid, the user can use the method "isPDOOutValid".

```
void sensorTask(){  
    deviceData=120; //Simulated sensor value  
    if(!sensorReady)  
        arduinoIOLink.controlPDIn(false);  
    else arduinoIOLink.controlPDIn(true);  
    uint8_t masterData;  
    arduinoIOLink.getPDOOut(&masterData);  
    if(arduinoIOLink.isPDOOutValid())  
        //Do something with master data  
}
```

Figure 24 Code snippet of user function for each cycle process

IODD GUI generator design and development

GUI Design

The necessity of a tool capable of modifying intuitively the framework parameters and reducing developing time brought the incorporation of a GUI into the project.

Furthermore, the advantages of the implementation of a GUI according to [12] are:

- Symbols are recognized faster than text
- Faster learning
- Faster use
- Exploits visual cues
- Fewer errors

The requirements for the GUI are as follows (Figure 25):

- Creation of an IODD file
- Creation of a configuration file for the IO-Link firmware
- Save and load feature

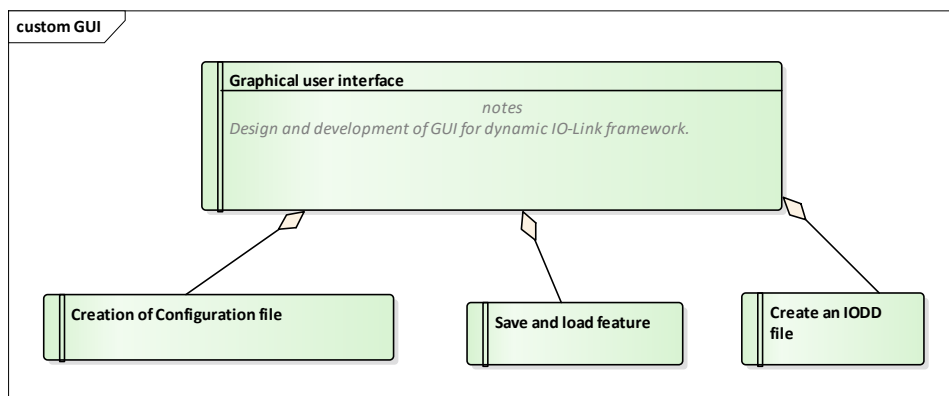


Figure 25 Requirements diagram for the GUI

The use case model presented in Figure 26 shows the functionality of the whole GUI application and how the interaction with the user shall be implemented.

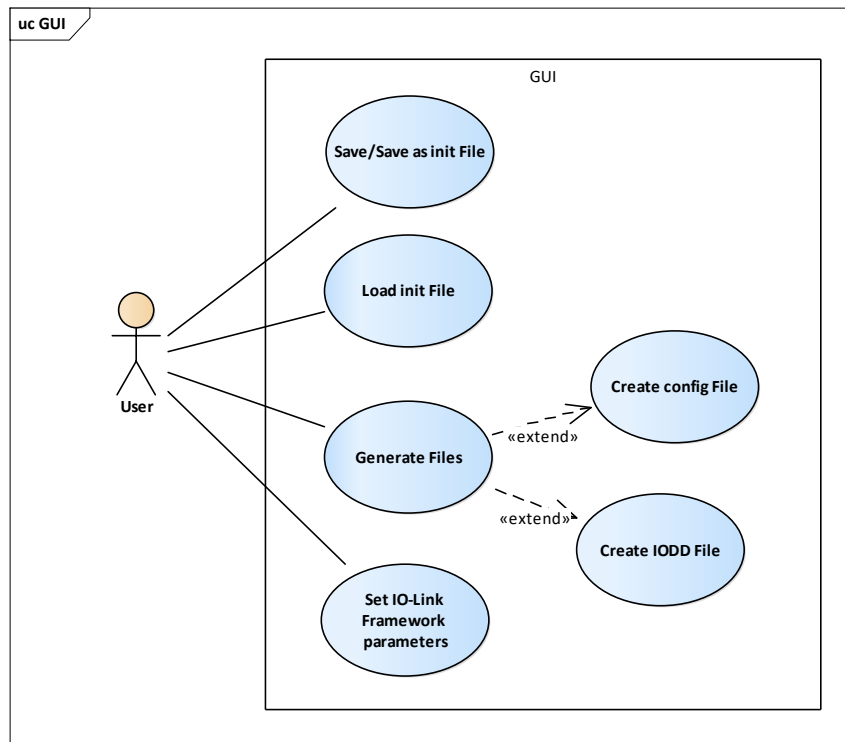


Figure 26 Use case diagram for the GUI based on the requirements

Use case	Description
Save/Save as init File	Save/Save as Init file with user-defined values
Load init File	Load init file with user-defined values
Generate Files	Creation of output files
Create Config File	Create header file used for firmware compilation
Create IODD File	Create IODD File with user-defined parameters

<p>Set IO-Link Framework parameters</p>	<p>Set parameters IO-Link parameters:</p> <ul style="list-style-type: none"> • Identification parameters • Process data structure • Custom Parameters • Events
--	--

Table 4 Use case description of GUI

Requirements of the configuration file generator

The configuration file for the IO-Link firmware has the appropriate initial values that the device should load when booting up. Since these parameters are stored in a header file it turns out to be quite tedious to modify this file each time. The purpose of including this task in a GUI is to make the process of writing the configuration file as simple as possible. Figure 27 shows the specific requirements that this task should meet.

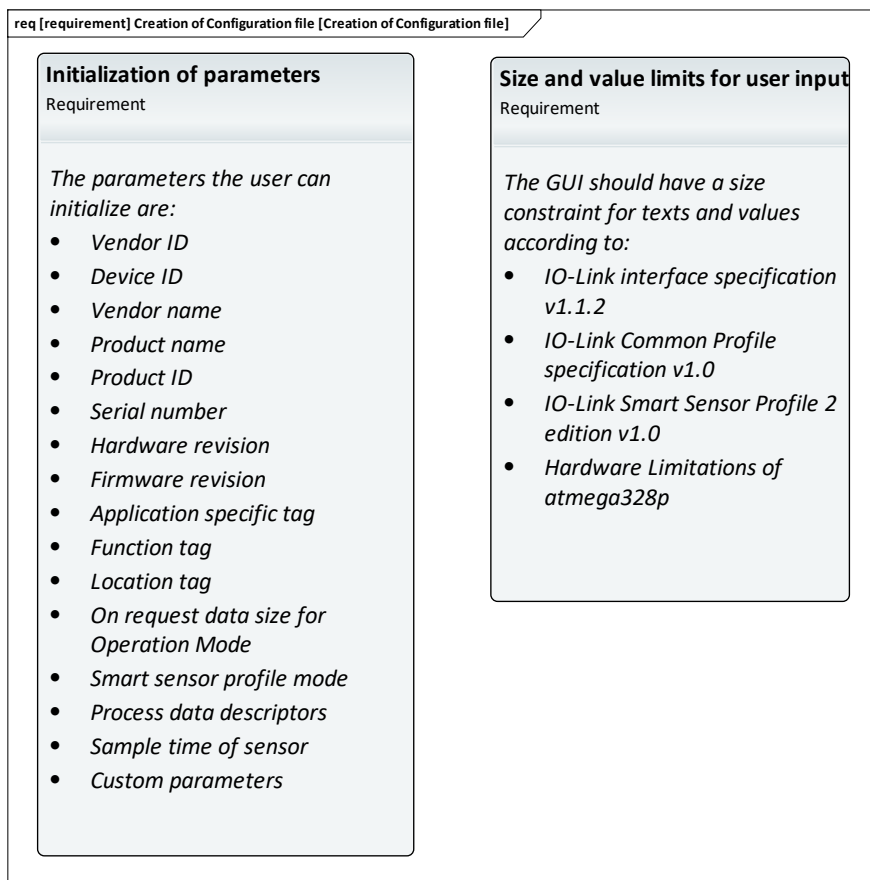


Figure 27 Requirements for the Creation of the configuration file

Requirements of the IODD file generator

The creation of the IODD file is essential for the device since it describes its parameters which can be later modified by an IODD tool. These parameters are defined according to the user's device and should follow the IO-Link IODD specification, the specific requirements for this task are shown in Figure 28.

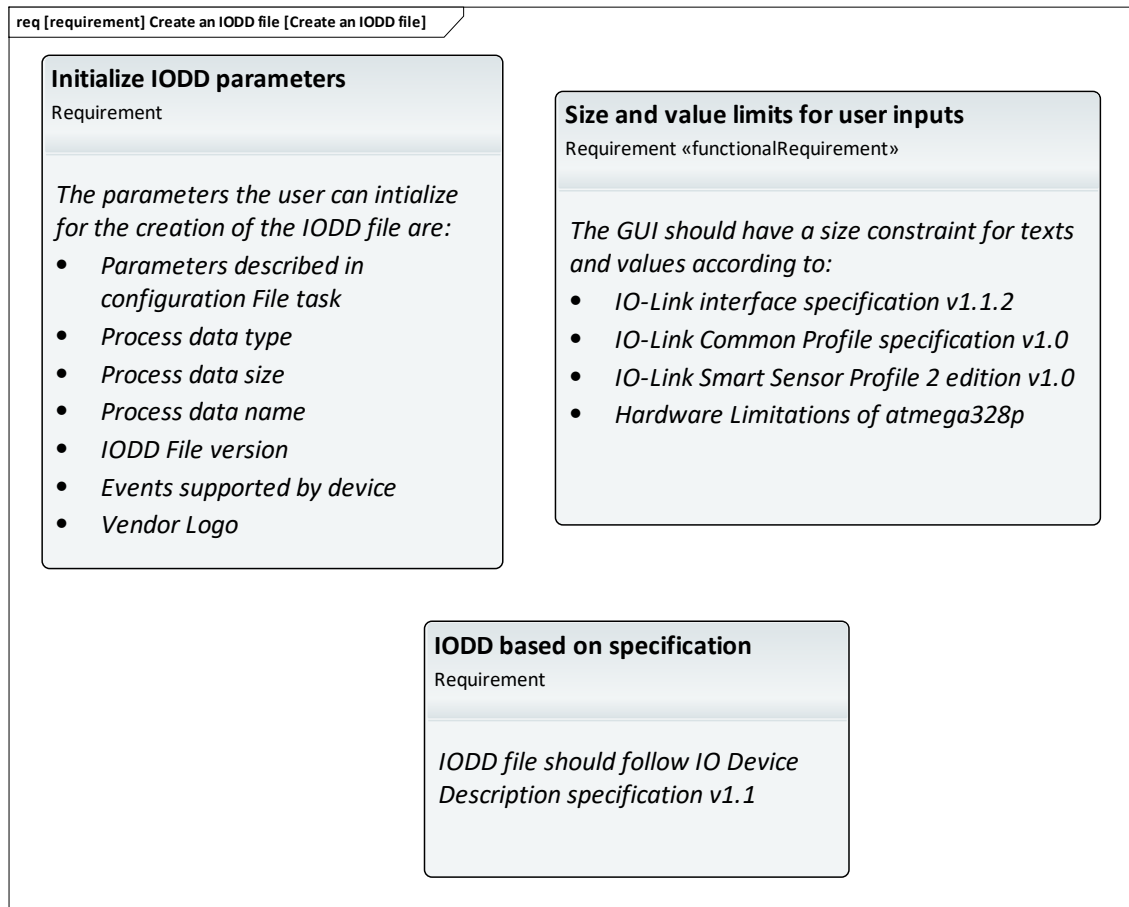


Figure 28 Requirements for the creation of the IODD file

Requirements of save and load feature

The purpose of this function is to reduce the time the user needs to configure a new or similar device which incorporates more or fewer features. The requirements diagram for this task is shown in Figure 29.

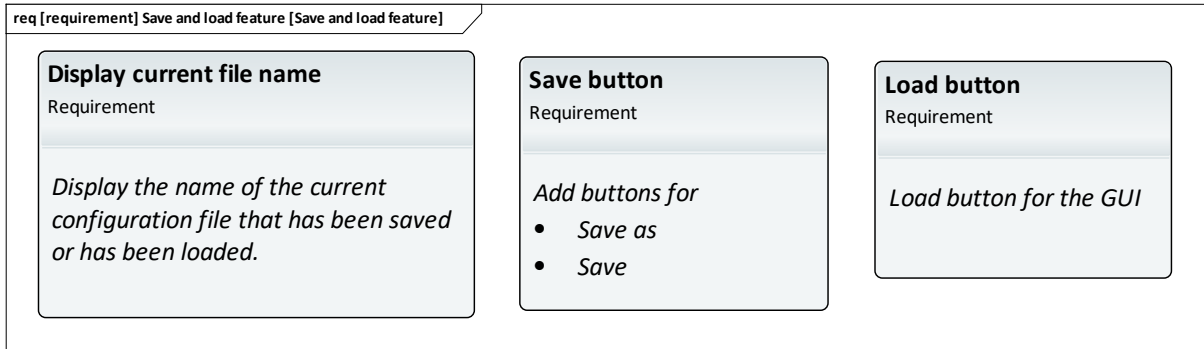


Figure 29 Requirements for Save and load feature task

Development of the GUI

Software dependencies

The development of this GUI was done with the Python interpreter (v3.6) programming language and the aid of the PyQt 5 library. PyQt is a set of bindings for the Qt application framework, which its purpose is the creation of graphical user interfaces capable of running on Windows, OS X, Linux, iOS and Android [13].

The GUI is based on the library PyQt 5, which has all the required elements for developing a complete GUI with abstract classes so the programmer only needs to focus on the design. Additionally, the software Qt Designer was used to manually set the location of objects, buttons, and widgets inside the GUI instead of coding each single GUI element.

File dependencies

The program includes a file directory with files occupied internally, which consist of:

- **File templates**
These files are related to the configuration file and Qt Designer file.
- **IODD Checker**
Software used for checking the validity of the IODD file
- **Icons**
Icons used for buttons and main window application

GUI overview

The GUI consists of one single window (see Figure 30) that allow the end-user to fill in the parameters for its application through the next tabs:

- **Identification parameters:** Device parameters that identify uniquely the user application.
- **Process Data:** Describes the type and size of data which is sent from the device to the master and vice-versa.
- **Custom parameters:** Parameters that a user can configure with the help of an IODD tool such as a PLC.
- **Events:** Indicates standard and custom events the device can support

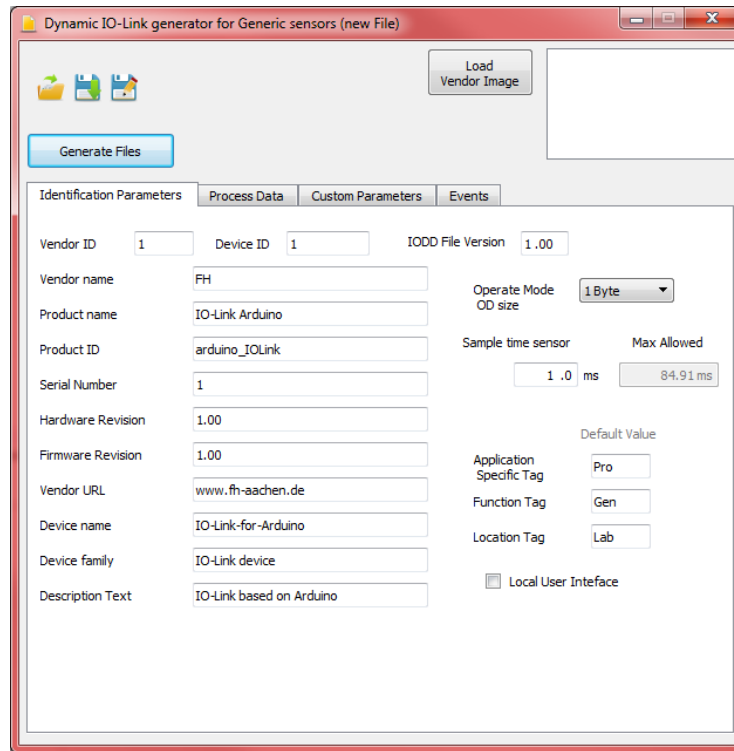


Figure 30 Main window of GUI

Additionally, from the Qt Designer software, it's possible to define size restrictions (see Figure 31). Hence, the values for strings of texts were constrained according to the IO-Link specification and hardware limitations within it.

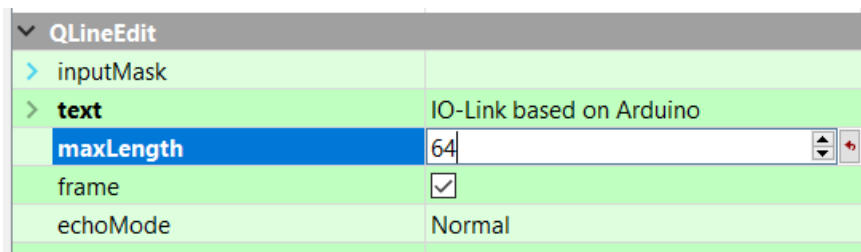


Figure 31 Qt Designer properties dialog

File generator

The generation of the files works by having the user fill the parameters its device will have and afterward, a button can be clicked to generate the files needed for the firmware and the IODD. Considering that the files need to comply with the IO-Link specification certain restrictions were implemented to avoid errors originated from a wrong input of the user.

The GUI notifies the user if the files cannot be created through a notification window that describes any specific error if it does not comply with the expected input or allowed range of values. The flowchart in Figure 32. shows an outline of how the verification system of user input works.

The way this error notification system works is by first checking if the identification parameters such as vendor ID and device ID are in the correct range. Subsequently, the process is checked if it's less than the maximum allowed size (32 bytes) and if both process data (output/input) are greater than 1 byte. Afterward, the smart sensor profile values are verified according to [11]. In addition, the custom parameters and events are checked if they are in range. Finally, if the cycle time is less than the maximum cycle time permitted in [3] then the program proceeds to create the files. Figure 33 shows some examples of how these error windows look like when they are activated.

Since the GUI also implements the feature of loading an image for the vendor, logo of the IODD file. The user can upload an image of any size and then the program will resize it to the size specified in [14]. If the image is not supported or is corrupted, it will not be loaded and referenced in the IODD file (see Figure 34).

The creation of the configuration file and sketch example for the framework are based on template files. These template files have defined strings, which serve as a cue to know where to write each parameter the user has defined in the GUI. Figure 35 shows a snippet of how these files look before and after the user modifies them with the GUI.

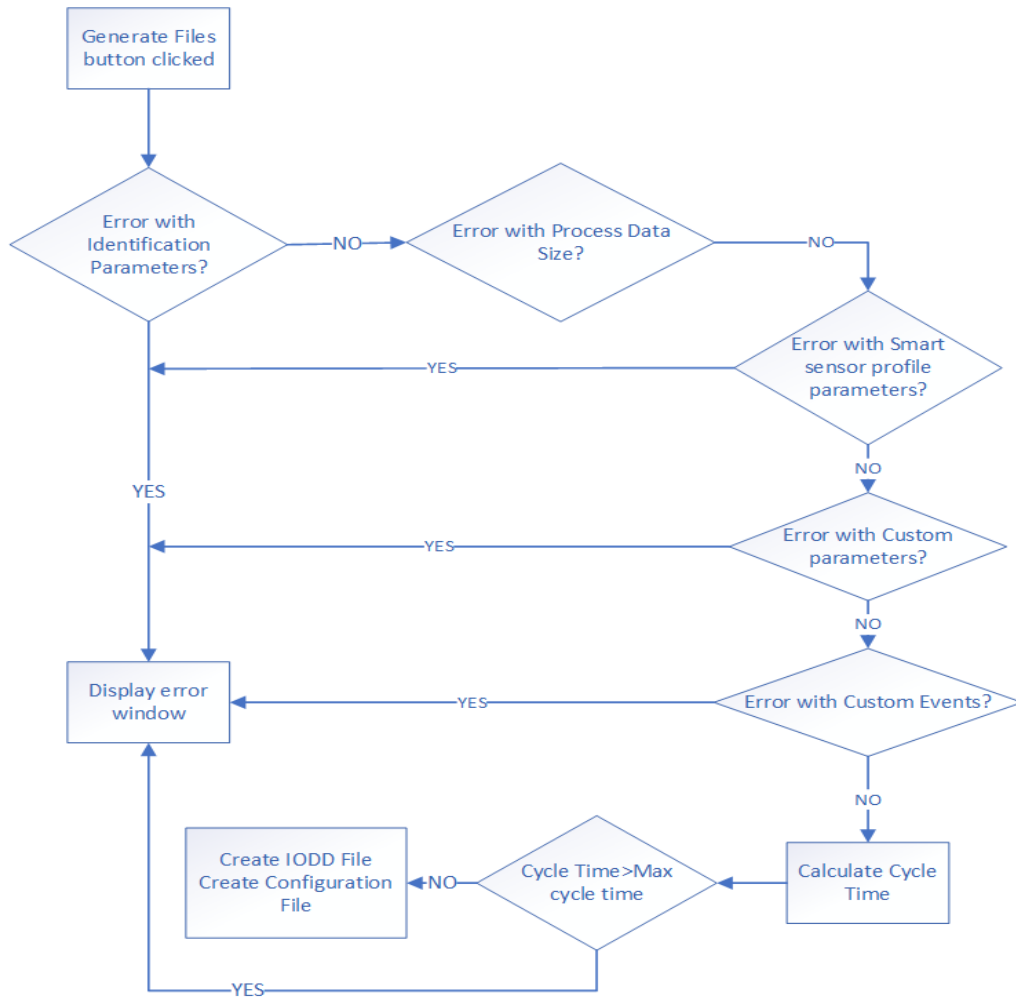


Figure 32 Flowchart for displaying errors when generating a file

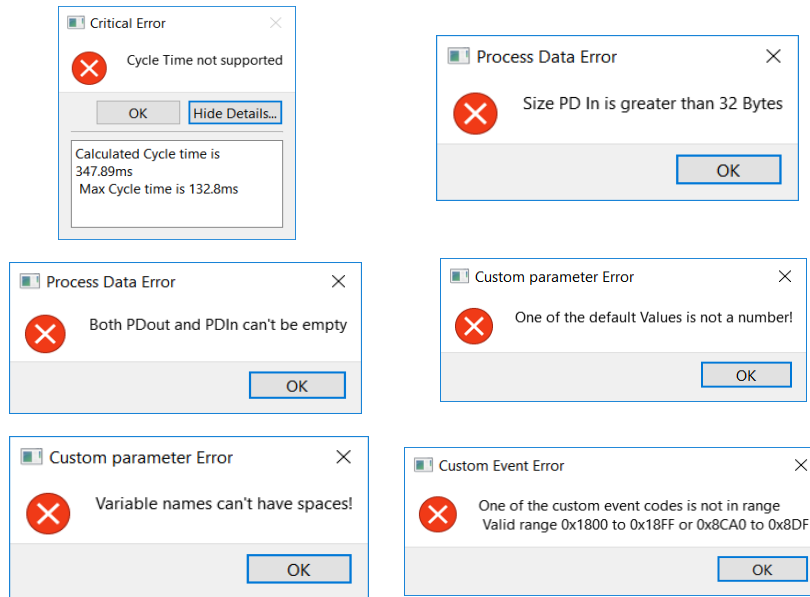


Figure 33 Examples of error notifications

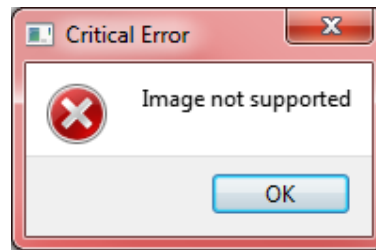


Figure 34 Image not supported error

```

/*
User Configuration file for IO-Link device options
*/
#ifndef CONFIG_H
#define CONFIG_H
/*-----
User defined parameters for specific application
-----*/
#define //Smart Sensor profile Mode
//-----
//Define Data storage if needed
#define MINCYCLETIME //Defined in B.1.3
//Defined in B.1.6
#define Length_PDOut //Min value 1
#define Length_PDIn //Min value 1
#define sizeOD_PREOPERATE 8 //Values should be 1,2,8
#define sizeOD_OPERATE //Values should be 1,2,8 or 3
#define specificTechParam //Number of specific technol
#define numberSpecificTechParamToSave //Number of cus
#define SPECIFIC_TECH_PARAM_SIZE_DS //Total size in o
#define sizeVendorName //Number of characters of vend
#define sizeProductName //Number of characters of Prod
#define VENDOR_ID //Default Vendor ID
#define DEVICE_ID //Default Device ID
/*-----*/
#if defined(FSS) || defined(AdSS) || defined(DMS) ||
#define DATASTORAGE_ENABLED
#define SMART_SENSOR_PROFILE //Needed for FSS,AdSS or
#define SIZE_APP_TAG //
#define SIZE_FUN_TAG //
#define SIZE_LOC_TAG //
#define APP_SPEC_TAG //Default App specific tag
#define FUNCTION_TAG //Default Function tag
#define LOCATION_TAG //Default Location Tag
#endif
*/
User Configuration file for IO-Link device options
*/
#ifndef CONFIG_H
#define CONFIG_H
/*-----
User defined parameters for specific application
-----*/
#define FSS//Smart Sensor profile Mode
#define FSS_SP 1000
//-----
//Define Data storage if needed
#define MINCYCLETIME 86//Defined in B.1.3 Cycle Time =
//Defined in B.1.6
#define Length_PDOut //Min value 1
#define Length_PDIn 1//Min value 1
#define sizeOD_PREOPERATE 8 //Values should be 1,2,8 or
#define sizeOD_OPERATE 8//Values should be 1,2,8 or 32
#define specificTechParam 0//Number of specific technolog
#define numberSpecificTechParamToSave 0//Number of custo
#define SPECIFIC_TECH_PARAM_SIZE_DS 0//Total size in oct
#define sizeVendorName 2//Number of characters of vendor
#define sizeProductName 15//Number of characters of Prod
#define VENDOR_ID 9999//Default Vendor ID
#define DEVICE_ID 1234//Default Device ID
/*-----*/
#if defined(FSS) || defined(AdSS) || defined(DMS) || def
#define DATASTORAGE_ENABLED
#define SMART_SENSOR_PROFILE //Needed for FSS,AdSS or DM
#define SIZE_APP_TAG 3//
#define SIZE_FUN_TAG 3//
#define SIZE_LOC_TAG 3//
#define APP_SPEC_TAG "Pro"//Default App specific tag
#define FUNCTION_TAG "Gen"//Default Function tag
#define LOCATION_TAG "Lab"//Default Location Tag
#endif

```

Figure 35 Snippet of configuration file template (left) and final output (right)

For the creation of the IODD file, the IODD specification requires to check the file with a proprietary software of the IO-Link Consortium group called IODD Checker, which can be found in [15]. If the file does not contain errors the IODD Checker, it writes a cyclic redundancy check (CRC) and a stamp code which indicates that the XML file complies with the IO-Link specification. Hence, as a final step, the IODD Checker is executed within the GUI program to comply with [14].

Furthermore, since the purpose of the firmware is to be used with the Arduino framework an additional task was implemented to create an example sketch with API reference so the final user can have an example for its application (see Figure 36). The general overview of the GUI program is described through the class diagram of Figure 37, where the relations between the objects that interact with the GUI and its methods are specified.

```
#include <device.h>

io_link_device arduinoIOLink; //IO-Link device
uint8_t deviceData[3]; //User must use this variable to update its sensor data
uint8_t masterData[1]; //User can use this variable to read data from master device

//Custom Parameters
uint8_t filter; //R-W variable Included in Data storage
uint16_t filter2; //R-W variable Included in Data storage

//User task for reading sensor and updating deviceData Variable
void sensorTask(){
    arduinoIOLink.getPDOOut(masterData); //Use this method to update PDOOut data from master device
}

void setup() {
    //Add User Parameter Variables
    arduinoIOLink.addParameter(&filter, sizeof(filter), true);
    arduinoIOLink.addParameter((uint8_t *)(&filter2), sizeof(filter2), true);

    arduinoIOLink.initDevice(sensorTask, deviceData); //Init IO-Link Device Settings
}

void loop() {
    arduinoIOLink.ioLink_Task();
}
```

Figure 36 Example of Sketch file for API reference

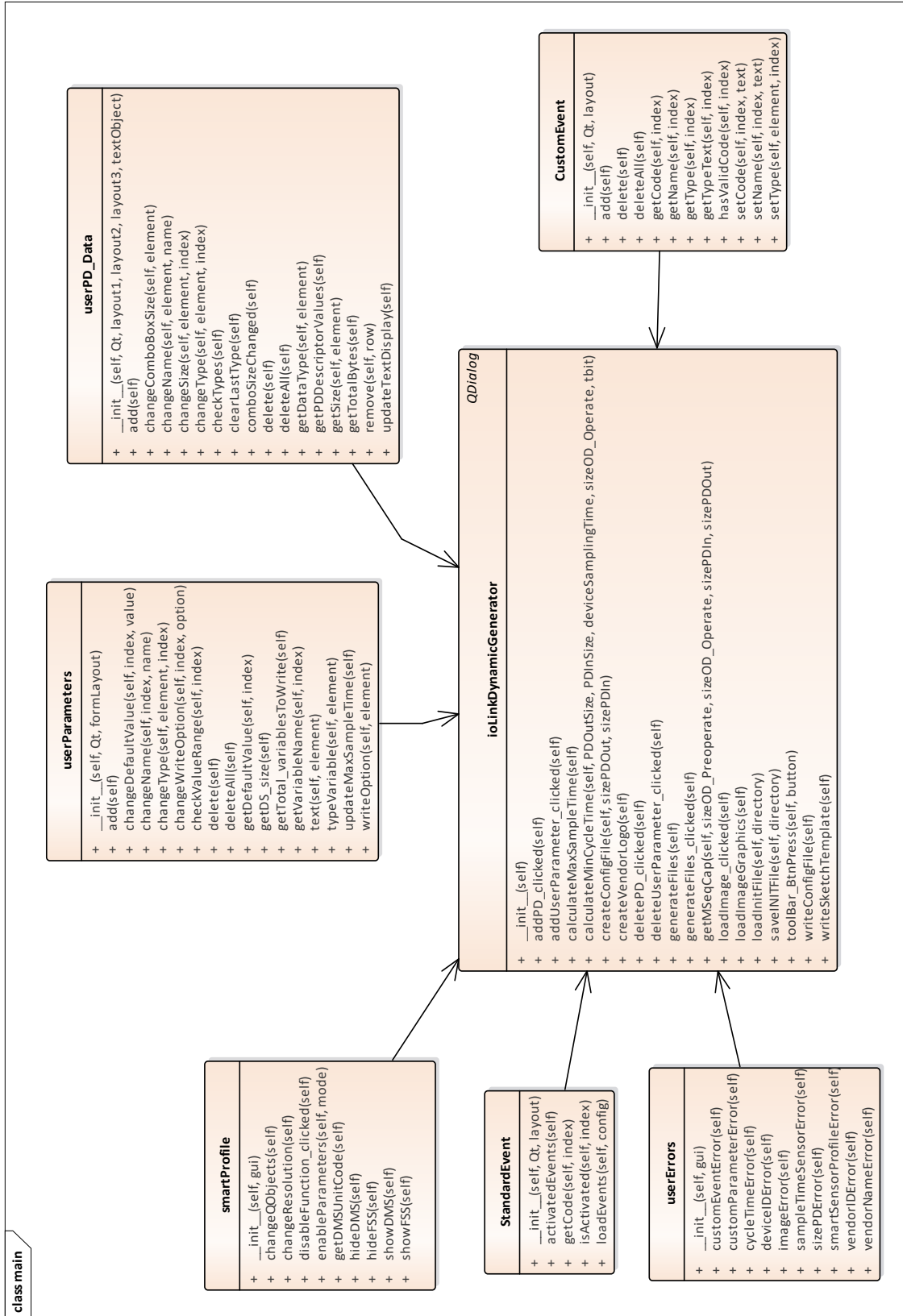


Figure 37 GUI Class diagram

GUI operation example

In this section, a practical example demonstrates the use of the GUI for the creation of an IODD file and the setup file for the IO-Link firmware. A full description of the GUI is available in Annexure 4 GUI user manual.

The setup for this example consists of a potentiometer whose value from the analog-to-digital converter (ADC) of the microcontroller is sent to the IO-Link master (see Table 5).

Characteristics	Description	
Process input data (2 Bytes)	ADC value from the potentiometer (2 Bytes)	Range of ADC is from 0 to 1023
Device specific parameters	Enable (1 Byte)	Enable sensor ON=1,OFF=0

Table 5 IO-Link potentiometer description

The first step is to open the GUI, in the identification parameters tab the related information from the vendor is set (e.g. name of the vendor, product name). In addition, the sample time of the sensor is set in this tab, for this example, it is assumed that the sensor task takes 1 millisecond for this device application (see Figure 38).

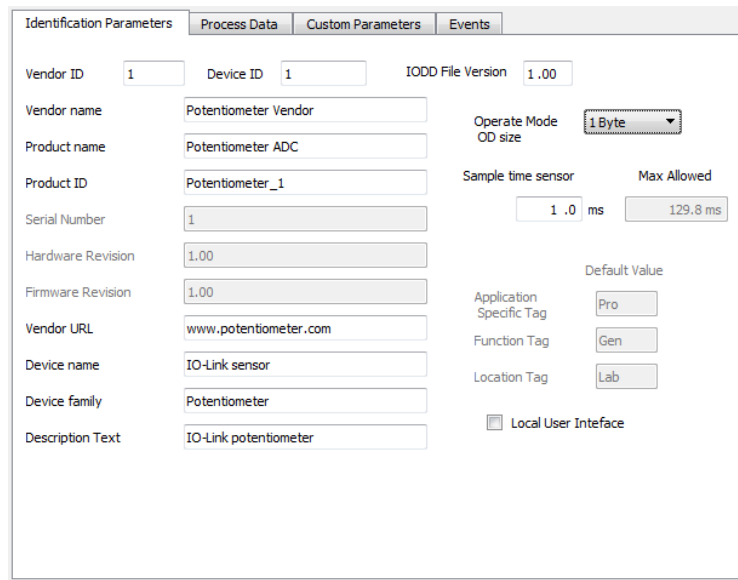


Figure 38 Identification parameters tab for the potentiometer device

The next step consists of setting the process data structure for the IO-Link device through the “Process Data” tab. For this example, a value of type unsigned integer and size of 2 bytes is selected (see Figure 39).

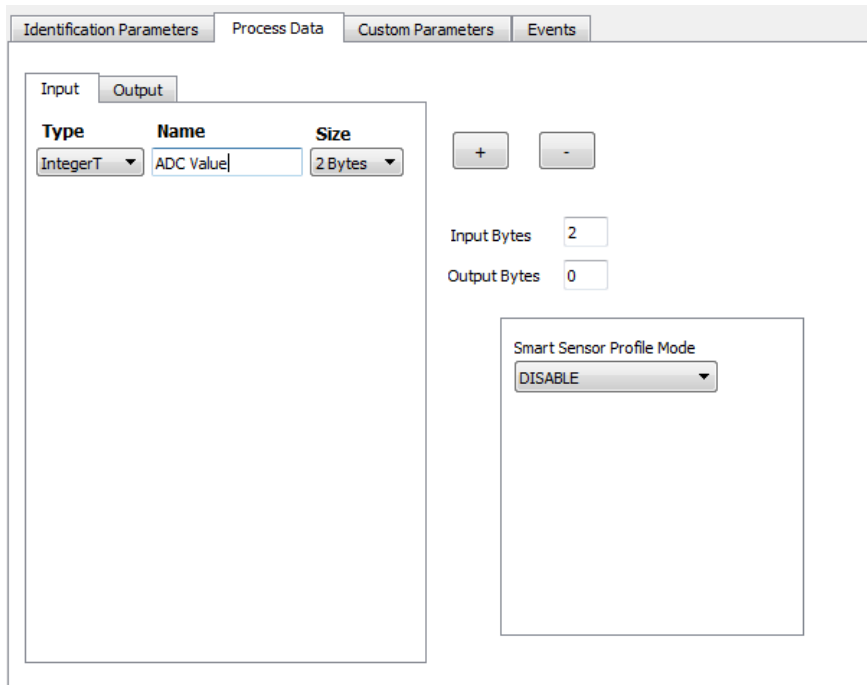


Figure 39 Process data tab for potentiometer device

Since the device has one custom parameter, it must be set on the “Custom Parameters” of the GUI. The parameter will be of type uint8_t, with write access and a default value of 1 (see Figure 40).

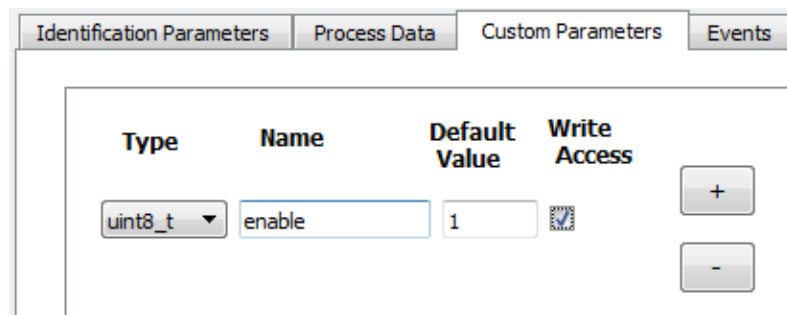


Figure 40 Custom Parameters tab for potentiometer device

Afterward, the end-user may save the current IO-Link device settings by clicking the button “Save” or “Save as” for future work or modifications. As a final step, the IODD file and the header file are generated by clicking the “Generate Files” button. Once the files have been generated, the file explorer opens the folder where they are saved (see Figure 41).

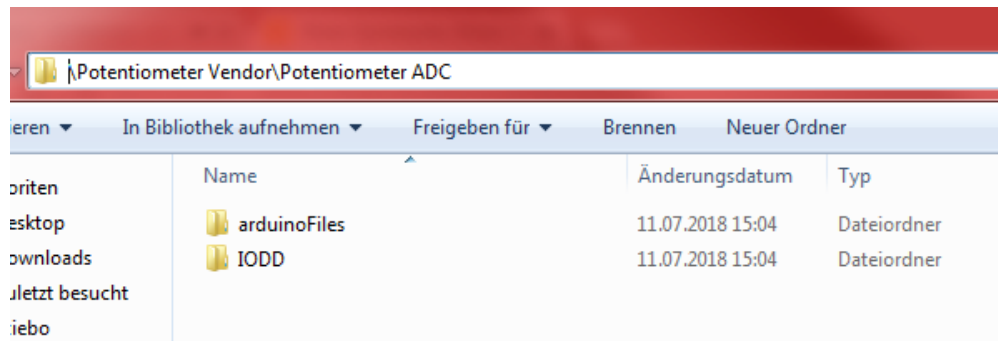


Figure 41 File explorer showing the GUI folder output

The folder “IODD” contains the XML file with the description of the device (see Figure 42). Additionally, the folder “arduinoFiles” contains the header file (see Figure 43), which must be compiled with the IO-Link firmware, and a sketch file (see Figure 44) which presents an example program of how to use the API for its device application.

```

~/results/
<VariableCollection>
  <StdVariableRef id="V_DirectParameters_1"/>
  <StdVariableRef id="V_DirectParameters_2"/>
  <StdVariableRef id="V_DeviceAccessLocks"/>
  <StdVariableRef id="V_SystemCommand"/>
  <StdVariableRef defaultValue="Potentiometer Vendor" id="V_VendorName"/>
  <StdVariableRef defaultValue="Potentiometer ADC" id="V_ProductName"/>
  <StdVariableRef defaultValue="Potentiometer_1" id="V_ProductID"/>
  <StdVariableRef defaultValue="1.00" id="V_HardwareRevision"/>
  <StdVariableRef defaultValue="1.00" id="V_FirmwareRevision"/>
  <StdVariableRef id="V_ProcessDataInput"/>
  <Variable accessRights="rw" defaultValue="1" id="V_enable" index="64">
    <Datatype bitLength="8" xsi:type="UIntegerT"/>
    <Name textId="TN_V_enable"/>
  </Variable>
</VariableCollection>
<ProcessDataCollection>
  <ProcessData id="PD">
    <ProcessDataIn bitLength="16" id="PDIn">
      <Datatype bitLength="16" subindexAccessSupported="false" xsi:type="RecordT">
        <RecordItem bitOffset="0" subindex="1">
          <SimpleDatatype bitLength="16" xsi:type="IntegerT"/>
          <Name textId="TN_PDIn_0"/>
        </RecordItem>
      </Datatype>
      <Name textId="TN_PDIn"/>
    </ProcessDataIn>
  </ProcessData>
</ProcessDataCollection>

```

Figure 42 Snippet of IODD file of the potentiometer device

```

/*
  User Configuration file for IO-Link device options
  */
#ifndef CONFIG_H
#define CONFIG_H
/*-----
User defined parameters for specific application
-----*/
#define DISABLE//Smart Sensor profile Mode
//-----
//Define Data storage if needed
#define MINCYCLETIME 97//Defined in B.1.3 Cycle Time = 19.6ms
//Defined in B.1.6
#define SIZE_PDOUT 0//Min value 1
#define SIZE_PDIN 2//Min value 1
#define SIZE_OD_PREOPERATE 8 //Values should be 1,2,8 or 32
#define SIZE_OD_OPERATE 1//Values should be 1,2,8 or 32
#define TOTAL_CUSTOM_PARAMETERS 1//Number of specific technology parameters
#define TOTAL_CUSTOM_PARAMETERS_WRITE 1//Number of custom parameters to save
#define SIZE_CUSTOM_PARAMETERS_WRITE 1//Total size in octets of custom param
#define DEFAULTVALUE_CUSTOM_PARAMETERS 1//Default Values to load for user pa
#define SIZE_VENDOR_NAME 20//Number of characters of vendor Name Max 64
#define SIZE_PRODUCT_NAME 17//Number of characters of Product Name Max 64
#define VENDOR_ID 1//Default Vendor ID
#define DEVICE_ID 1//Default Device ID

```

Figure 43 Snippet of header file of the potentiometer device


```

#include <IO-Link Device.h> //IO-Link library

io_link_device arduinoIOLink; //IO-Link device
uint8_t deviceData[2]; //User must use this variable

//Custom Parameters
uint8_t enable;//R-W variable Included in Data storage

//User task for reading sensor and updating deviceDat
void sensorTask(){

}

void setup() {
//Add User Parameter Variables
arduinoIOLink.addParameter(&enable,sizeof(enable),fal

arduinoIOLink.initDevice(sensorTask,deviceData); //In

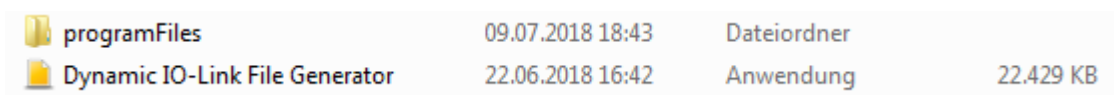
```

Figure 44 Snippet of sketch example for the potentiometer device

Software distribution

The final step of the development consisted of choosing the way the software would be distributed to the end-user. Since the software is based on the python language and needs external dependencies each time it is installed on an operating system, it made sense to compile the dependencies and the python interpreter in a single file with the source code.

The operating system that would be used for the distribution of the program was Windows since one of the software dependencies that it uses (i.e. IODD checker v1.4) is only distributed for Windows systems. For this purpose, the software “Pyinstaller” was used. This software allows compiling the source code of python program and all its python dependencies in a single executable File. Figure 45 shows the file structure of the software distribution for Windows-based systems.



programFiles	09.07.2018 18:43	Dateiordner	
Dynamic IO-Link File Generator	22.06.2018 16:42	Anwendung	22.429 KB

Figure 45 File structure for final distribution of GUI

Hardware design and development

The hardware components of the framework are the IO-Link transceiver and the microcontroller. Since there are already development board kits that include the atmega328p microcontroller for the Arduino framework only the IO-Link transceiver with the appropriate setup was designed.

The software used for the design of this project is EAGLE from Autodesk. EAGLE is capable of the design of electronic schematics, libraries, footprints for integrated circuits, printed circuit board designs and computer-aided manufacturing files.

The transceiver used for the physical communication layer is the TIOL111-5 from Texas Instruments; the manufacturer indicates that the following considerations should be taken in to account for the correct operation of the device:

- Each digital TTL output (WAKE and NFAULT) should have a 10 K Ω pull-up resistor.
- Minimum 100 nF capacitor between L+ and L-
- Minimum 1 μ F capacitor between VCC_IN/OUT and L-

As [3] indicates, the maximum current an IO-Link master can deliver to an IO-Link device is 200 mA. Due to the fact that the TIOL111-5 chip has a maximum output current of 300 mA, a resistor of 15k Ω is connected to pin ILIM_ADJ to limit the current approximately to 200mA as specified in [9]. In addition, the transceiver's VCC_IN pin has a 5V output, so it's used as the power supply of the atmega328p. Furthermore, an M12 class A male connector was added to comply with the electric input connection specified in [3].

Another point to consider for the design is that when the Arduino board is programmed its internal voltage regulator is powered up and its output is connected to the low-dropout (LDO) regulator output of the transceiver. A Schottky diode is implemented as input protection for the LDO regulator of the transceiver because it may be damaged if it receives a reverse polarity voltage from the Arduino board.

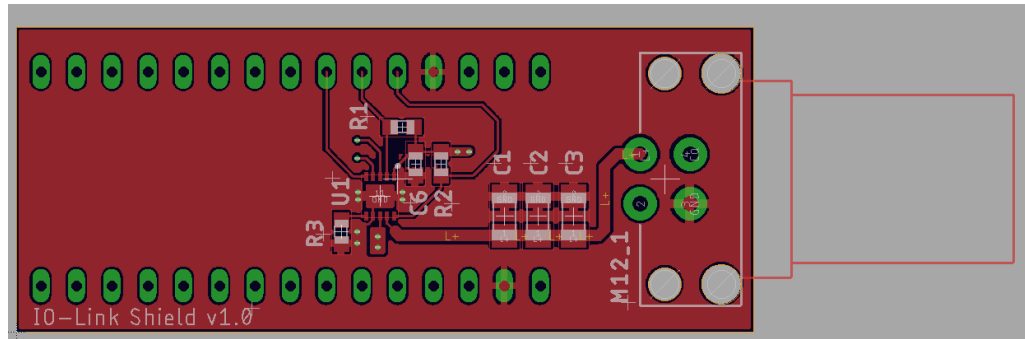


Figure 47 Top layer PCB preview

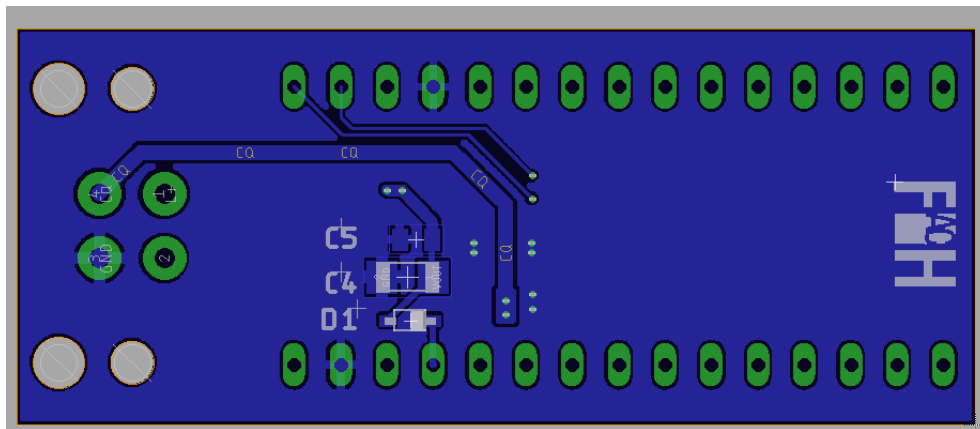


Figure 48 Bottom layer PCB preview

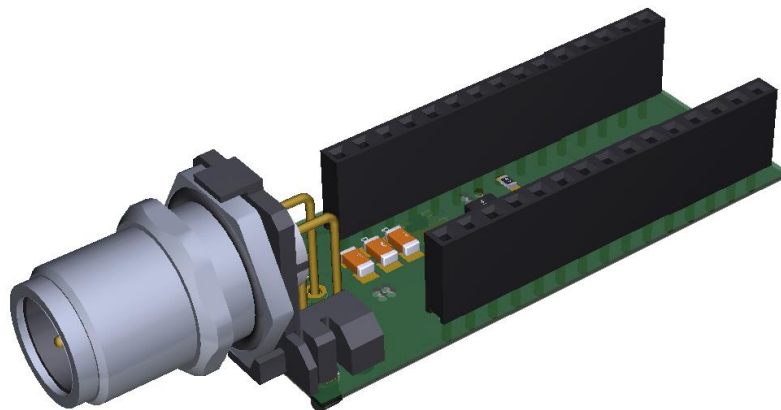


Figure 49 3D preview of PCB design

Results

This section provides the implementation of the framework with different examples. All of the tests included the use of the following hardware:

- IO-Link transceiver PCB
- WAGO 4-Channel IO-Link Master 750-657
- WAGO PLC 750-8206
- Arduino NANO (atmega328p) development board

Joystick with LED control

Device description

This example consists of a joystick with two analog axes output and the control of an RGB LED (see Figure 50). Further information about the general description of this device is shown in Table 7.

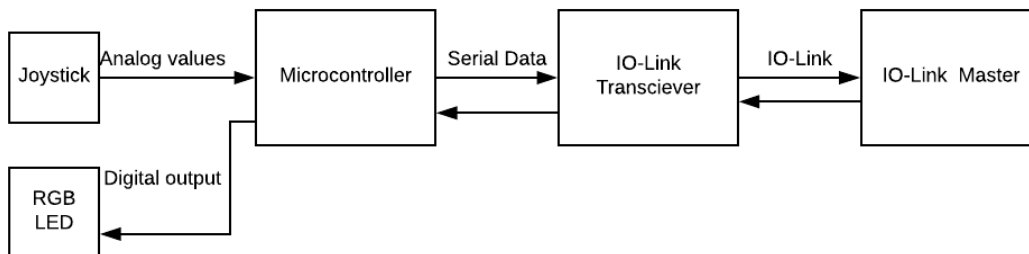


Figure 50 Data flow for joystick example

Characteristics	Description	
Process input data (16 bits)	Axis 1	8-bit Value (0-100)
	Axis 2	8-bit Value (0-100)
Process output data(8 bits)	Control	Controls the state of the current selected LED. 1=ON 0=OFF
	Device specific parameters	Color

Table 7 Joystick with LED control device description

IODD file generation

The IO-Link device is created with the help of the GUI (see Figure 51). The parameters that describe the device are located in the Process data Tab and Custom Parameters Tab as seen in Figure 52 and Figure 53 respectively. Once these parameters are set, the IODD file and the configuration file are generated.

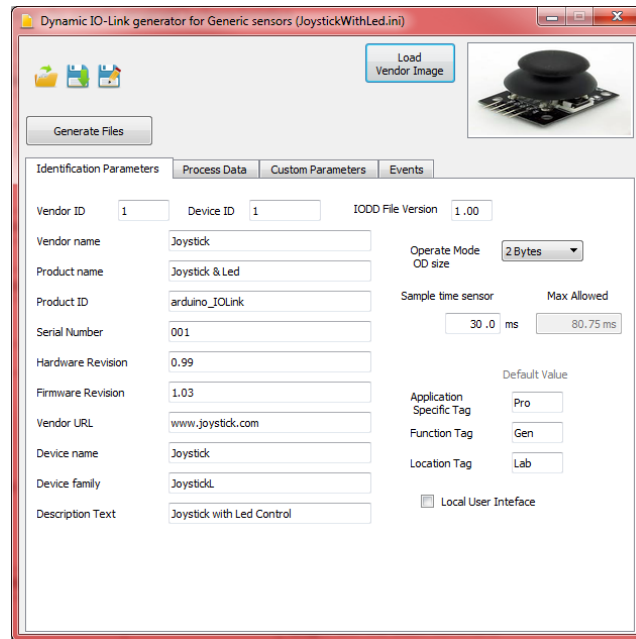


Figure 51 Identification Parameters in GUI for example 1

Type	Name	Size
UIntegerT	Axis 1	1 Byte
UIntegerT	Axis 2	1 Byte

Input Bytes: 2
Output Bytes: 1

Smart Sensor Profile Mode: GENERIC_SENSOR

Figure 52 Process data Tab input for Joystick device

Type	Name	Default Value	Write Access
uint8_t	color	1	<input checked="" type="checkbox"/>

Figure 53 Custom parameters Tab input for joystick device

Arduino sketch

The program for the Arduino Framework is shown in Figure 54. The program consists of reading the process output data from the IO-Link master and reading the custom parameter “color” to know which digital output from the microcontroller should be controlled.

Afterward, the program is compiled and tested with an IO-Link Master. The first test is to read the IO-Link device parameters. As seen in Figure 55 the identification parameters tab set in the GUI are the same and indicate that the framework is operating correctly. Consequently, the custom parameter “color” is changed to control a specific digital output for the RGB LED (see Figure 56).

```
#include <IO-Link Device.h>

io_link_device arduinoIOLink;
uint8_t deviceData[2]; //User must use this variable to update its sensor data
uint8_t masterData[1]; //User can use this variable to read data from master device
//Digital output LEDs declaration
uint8_t pinGreen =6;
uint8_t pinRed=5;
uint8_t pinBlue=7;
//User Parameter Variables declaration
uint8_t color; //R-W variable Included in Data storage

void colorOutput(uint8_t g,uint8_t b,uint8_t r){
    digitalWrite(pinRed,r);
    digitalWrite(pinBlue,b);
    digitalWrite(pinGreen,g);
}

void sensorTask(){
    deviceData[0]=map(analogRead(0),0,1023,0,100);
    deviceData[1]=map(analogRead(1),0,1023,0,100);
    arduinoIOLink.getPDOOut(masterData);
    if(masterData[0]==1){
        if(color==1)
            colorOutput(1,0,0);
        if(color==2)
            colorOutput(0,1,0);
        if(color==3)
            colorOutput(0,0,1);
    }
    else{
        colorOutput(0,0,0);
    }
}

void setup() {
    pinMode(pinGreen,OUTPUT);
    pinMode(pinRed,OUTPUT);
    pinMode(pinBlue,OUTPUT);
    colorOutput(0,0,0);
    arduinoIOLink.addParameter(&color,sizeof(color),true);
    arduinoIOLink.initDevice(sensorTask,deviceData) //Init IO-Link Device Settings
}

void loop() {
    arduinoIOLink.ioLink_Task();
}
```

Figure 54 Arduino sketch for Joystick device

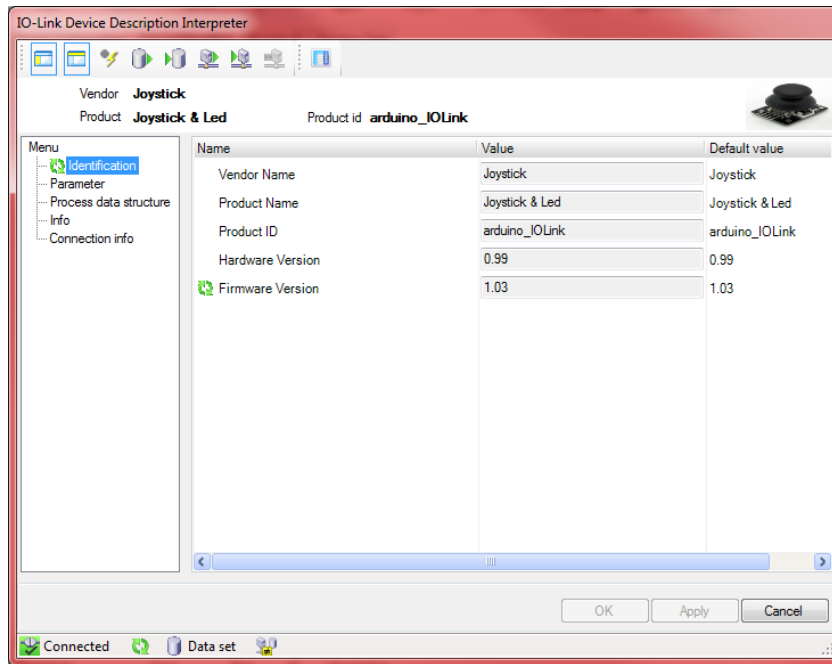


Figure 55 Validation of parameters for joystick device

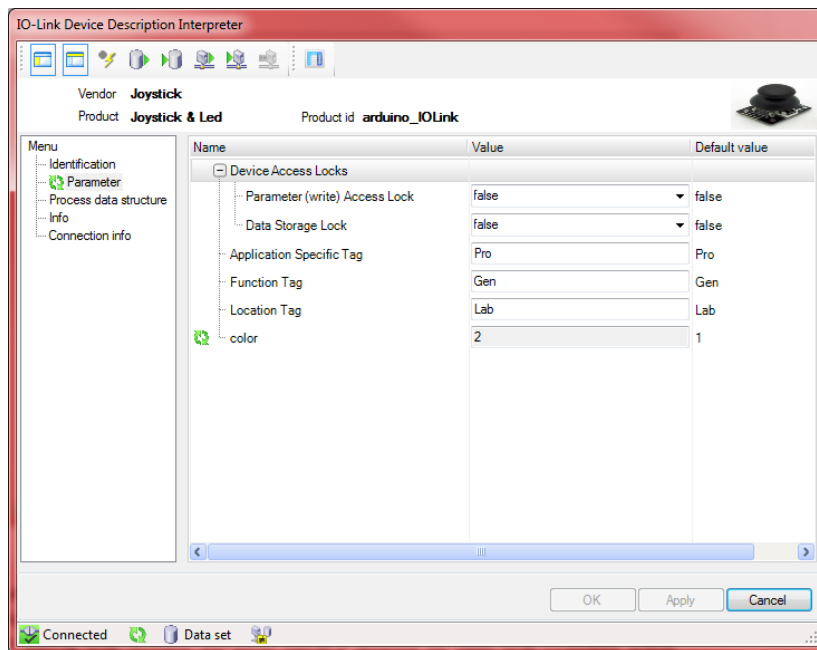


Figure 56 Changing custom parameter for joystick device

Gateway application

As a means of reading the IO-Link master information, a PLC acts as a gateway to test the values from the joystick. Figure 57 shows a small program using the WAGO IO-Link library that allows testing the IO-Link communication with the master module. From the WAGO function block, it can be seen that the number of bytes that it receives (i.e. variable udiReceievedBytes) from the device is 2, which corresponds to the same number as shown in Figure 52.

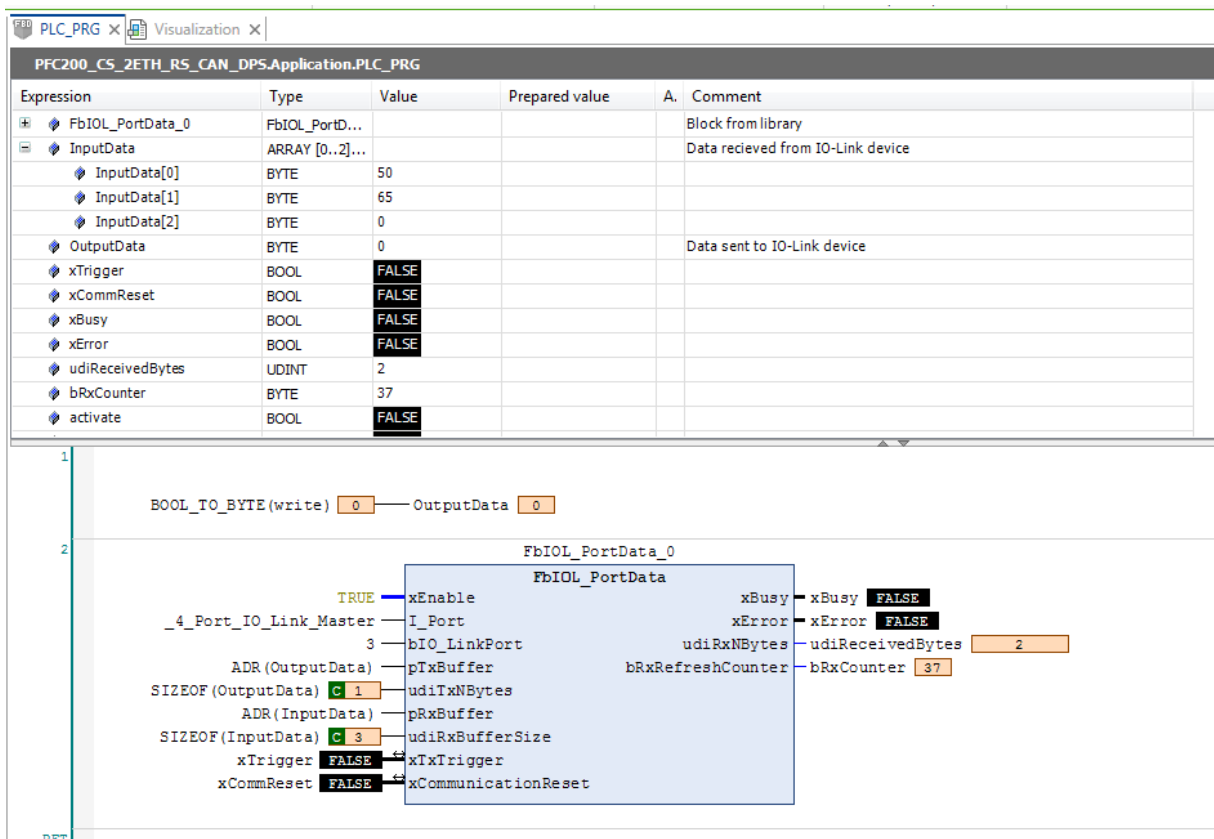


Figure 57 Snippet of PLC program for joystick device

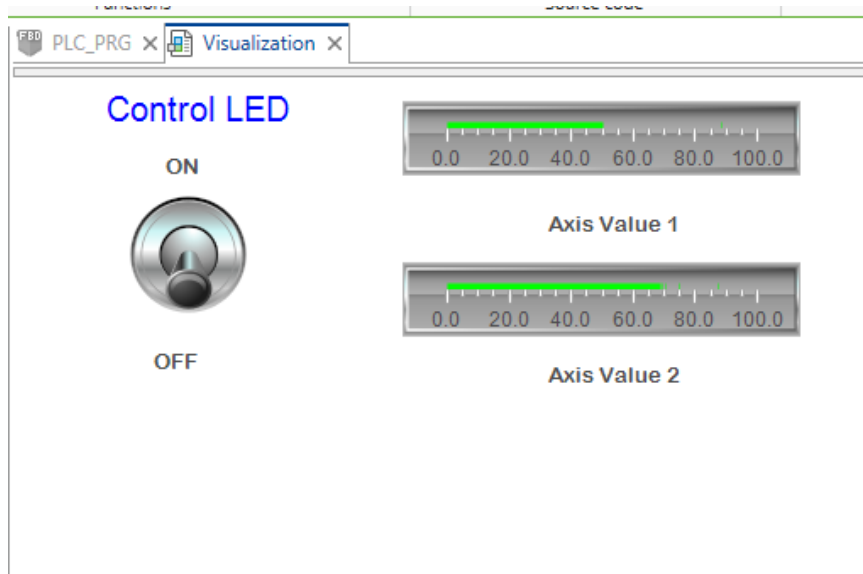


Figure 58 Reading axes information from PLC GUI

LED strip controller

Device description

In this example, data from the IO-Link master is sent to control a LED strip (see Figure 59). The master has control of the brightness of the LED strip, the number of LEDs (pixels) to control and individual control of each LED. Table 8 describes the behavior for this IO-Link device.

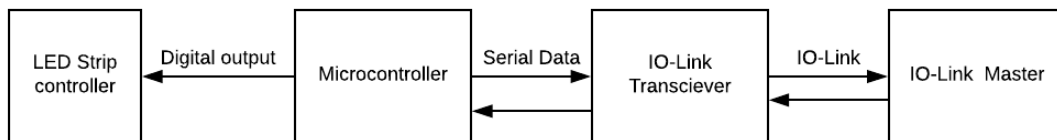


Figure 59 Data flow of LED strip controller

Characteristics	Description	
Process output data (12 bytes)	Pixel control (8bytes)	Control individual pixels in a LED strip (each bit value position controls one individual pixel) 1=ON 0=OFF
	Brightness (1 Byte)	General brightness of LED strip (1-255)
	Color R	Value for red value
	Color G	Value for Green value
	Color B	Value for Blue value
Device specific parameters	Number of pixels	Number of pixels in the LED strip (1 to 64)

Table 8 LED strip controller device description

IODD file generation

Accordingly, the GUI parameters are set to match the required aforementioned description (see Figure 60). This IO-Link device will not have a smart sensor profile, therefore, the “DISABLE” option is selected in the “Process data Tab” (see Figure 61). In addition, to control the number of pixels a custom parameter is added to the device (see Figure 62).

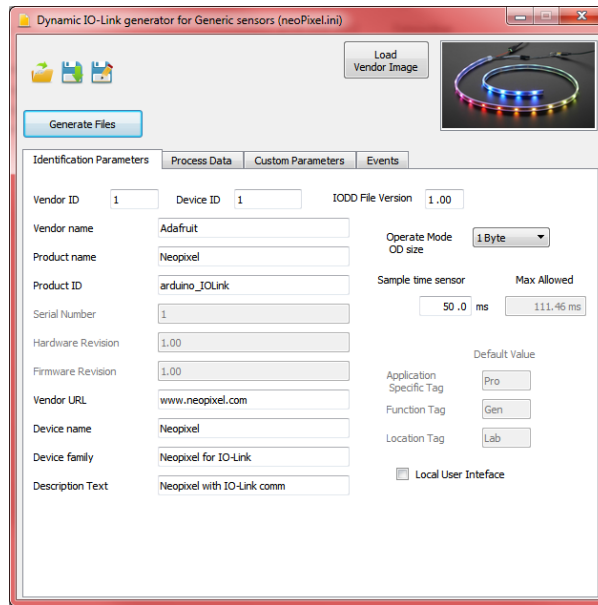


Figure 60 LED strip description in GUI

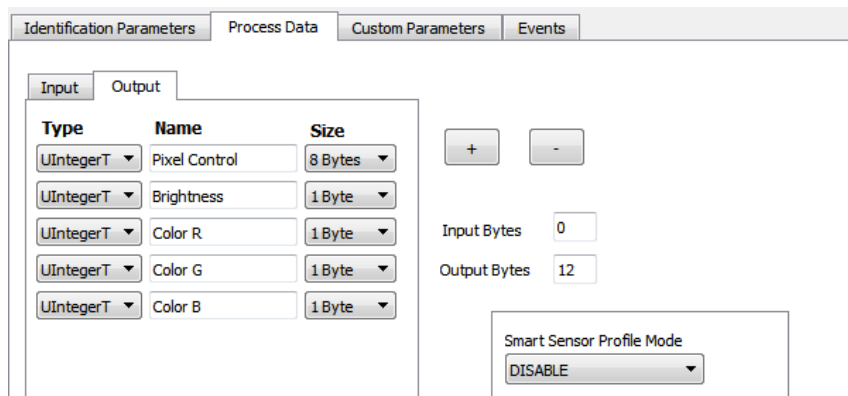


Figure 61 Process data tab for LED strip

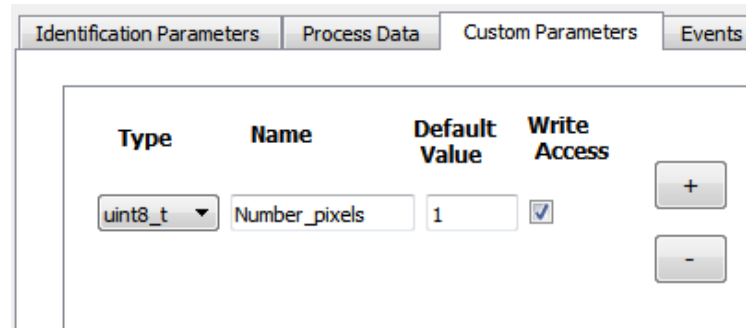


Figure 62 Custom parameters tab for LED strip

Arduino sketch

The main program for the Arduino framework (see Figure 63) reads the process output data from the master and then processes it. The first 8 bytes control each “pixel” connected to the LED strip, the next byte has the brightness value for the whole LED strip and the last three bytes indicate the RGB (red, green, blue) color for the LEDs.

```
#include <Adafruit_NeoPixel.h>
#include <IO-Link Device.h> //IO-Link library
#define PIN 7

io_link_device arduinoIOLink; //IO-Link device
uint8_t masterData[12]; //User can use this variable to read data from master device

//Custom Parameters
uint8_t Number_pixels; //R-W variable Included in Data storage

Adafruit_NeoPixel pixels = Adafruit_NeoPixel();
bool isPixelEnabled(uint8_t pixelPosition) {
    uint64_t controlData = 0;
    for (int i = 0; i < 8; i++)
        controlData |= masterData[i] << (8 * i);
    return (controlData >> pixelPosition) & 1;
}
//User task for reading sensor and updating deviceData Variable
void sensorTask() {

    arduinoIOLink.getPDOOut(masterData); //Use this method to update PDOOut data from master device
    pixels.setBrightness(masterData[8]);
    for (int i = 0; i < Number_pixels; i++) {
        if (isPixelEnabled(i))
            pixels.setPixelColor(i, pixels.Color(masterData[9], masterData[10], masterData[11])); // Set RGB for
current LED
        else
            pixels.setPixelColor(i, pixels.Color(0, 0, 0));
    }
    pixels.show(); // This sends the updated pixel color to the hardware.
}

void setup() {
    arduinoIOLink.addParameter(&Number_pixels, sizeof(Number_pixels), true);
    arduinoIOLink.initDevice(sensorTask);
    pixels.begin(); // This initializes the NeoPixel library.
    pixels.updateLength(Number_pixels);
    pixels.updateType(NEO_GRB + NEO_KHZ800);
    pixels.setPin(PIN);
}

void loop() {
    arduinoIOLink.ioLink_Task();
}
```

Figure 63 Arduino sketch for LED strip

Gateway application

To validate the IO-Link device the parameter “Number_pixels” set in the GUI is overwritten in the IODD tool of the IO-Link master (see Figure 64). Then a WAGO PLC test program is used to send the process output data to the device and control the LED strip (see Figure 65 and Figure 66).

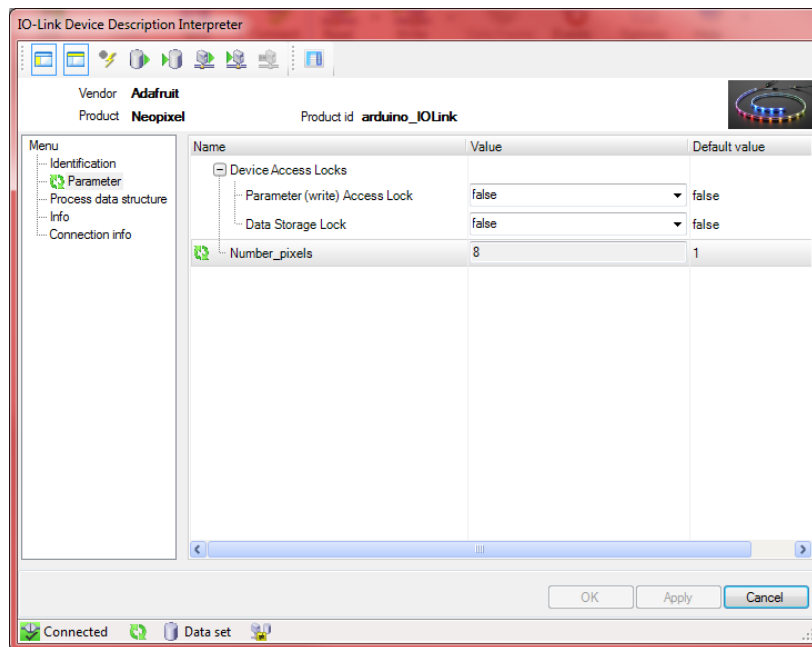


Figure 64 Reading and writing parameters to led strip device


```

PLC_PRG x Visualization x Library Manager x POU x
1 PROGRAM PLC_PRG
2 VAR
3 //Direct Call to Master to retrieve Data from Sensor
4 FbIOL_PortData_0: FbIOL_PortData; //Block from library
5 //Process Data
6 InputData: ARRAY [0..6] OF BYTE; //Data received from IO-Link device
7 xCommReset: BOOL;
8 xBusy : BOOL;
9 xError : BOOL;
10 udiReceivedBytes : UDINT;
11 bRxCounter : BYTE;
12 //GUI Variables
13 brightness:BYTE;
14 pixelControl: ARRAY [0..3] OF BOOL;
15 colorR:BYTE;
16 colorG:BYTE;
17 colorB:BYTE;
18 END_VAR
19 VAR_INPUT
20 xTrigger : BOOL; //Trigger Transmit from IO-Link master to device
21 OutputData: ARRAY [0..11] OF BYTE; //Data to send to IO-Link device
22 END_VAR
    
```

The visualization shows the variable declarations from the code above. A data block named 'FbIOL_PortData_0' is connected to the program. The connections are as follows:

- `xEnable` (TRUE) to `xEnable`
- `i_Port_IO_Link_Master` (1) to `i_Port`
- `ADR (OutputData)` to `pTxBuffer`
- `SIZEOF (OutputData)` to `udiTxNBytes`
- `ADR (InputData)` to `pRxBuffer`
- `SIZEOF (InputData)` to `udiRxBufferSize`
- `xTrigger` to `xTxTrigger`
- `xCommReset` to `xCommunicationReset`
- `xBusy` to `xBusy`
- `xError` to `xError`
- `udiReceivedBytes` to `udiReceivedBytes`
- `bRxCounter` to `bRxCounter`

Figure 65 PLC Program for the control of the LED strip

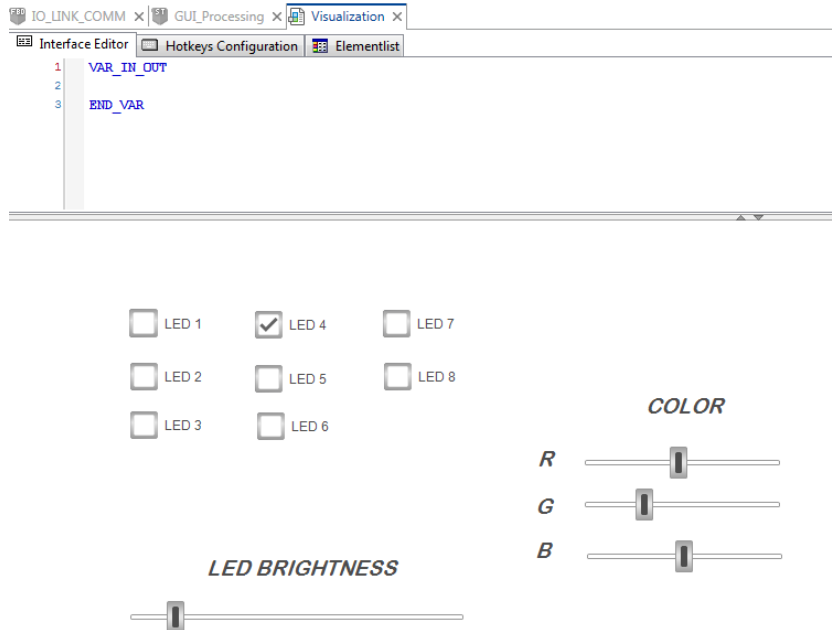


Figure 66 GUI for PLC program of LED strip control

Distance sensor

Device description

For this example, a distance sensor from Sharp model GP2Y0A21YK0F was implemented. This sensor has an analog output, which corresponds proportionally to the distance measured. For purposes of testing different functionalities of the framework, this sensor was implemented with the smart sensor profile for digital measuring sensors described in [11]. Table 9 describes the requirements for this IO-Link device.

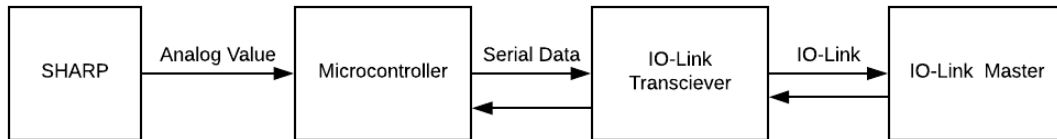


Figure 67 Data flow for distance sensor device

Characteristics	Description	
Process input data (4 bytes)	Vendor-specific byte (1 byte)	8-bit Value for the specific vendor application.
	Scale (1 byte)	Scale of the measurement value. i.e. Value*10 ^(scale)
	Measurement value (2 bytes)	16 bit value from sensor

Table 9 Distance sensor device description

The parameters of identification and process data were set to match the required sensor as seen in Figure 68 and Figure 69. In addition, since the sensor values reliability decrease if the object is too close or far from the object, the addition of diagnostic events was used (see Figure 70).

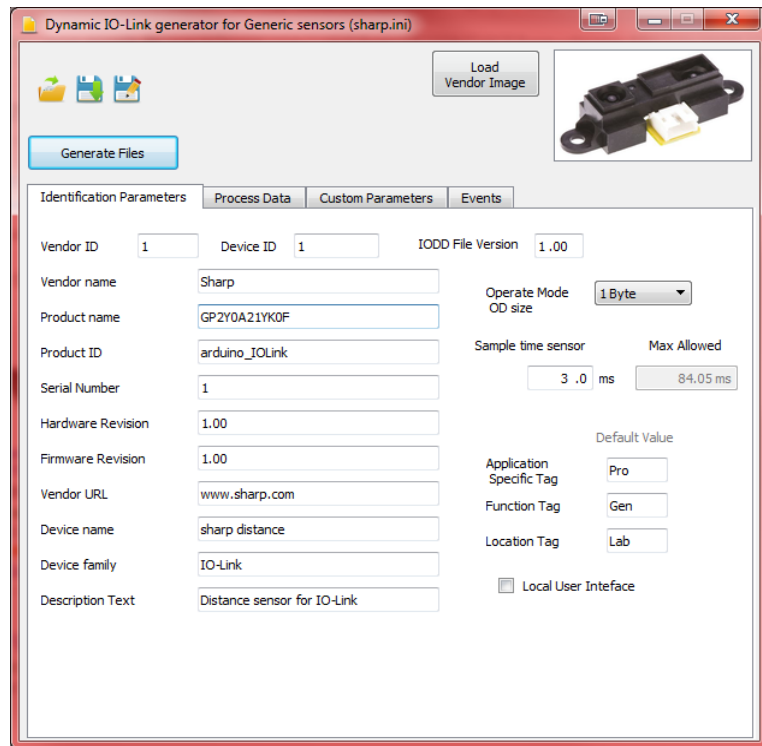


Figure 68 GUI Identification parameters for distance sensor

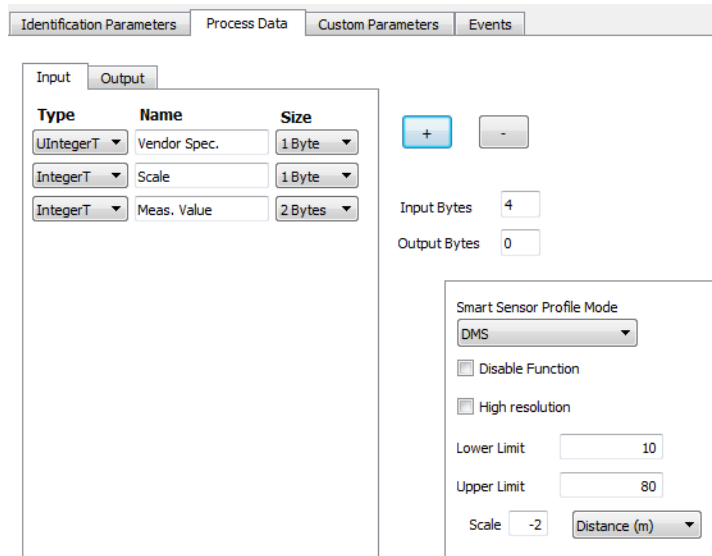


Figure 69 GUI Process data tab for distance sensor

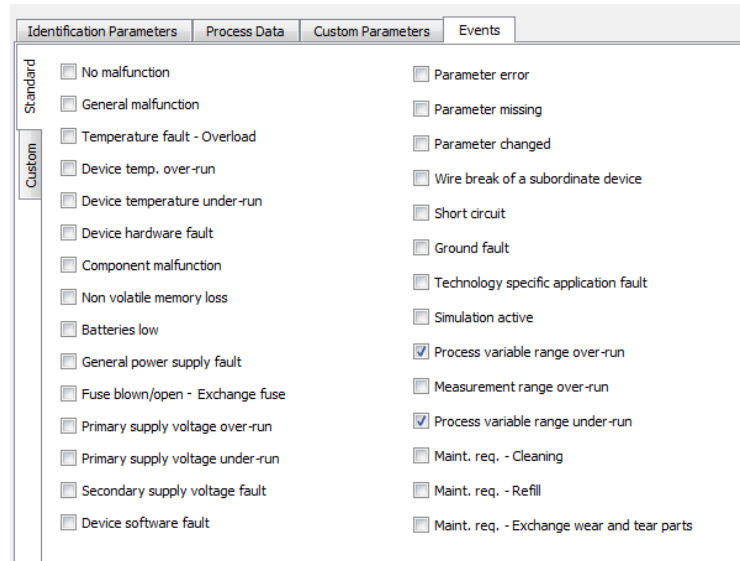


Figure 70 GUI Event tab for distance sensor

Arduino sketch

The main routine for the microcontroller consists of reading the analog value of the sensor and converting it to the distance measured in centimeters (see Figure 71). Due to the low range and far range limitation, standard events from the IO-Link specification were implemented so each time a certain threshold is overrun or underrun the IO-Link master can be notified (see Figure 72).

```
#include <IO-Link Device.h> //IO-Link library

io_link_device arduinoIOLink; //IO-Link device
uint8_t deviceData[4]; //User must use this variable to update its sensor data

IOLink_event measurementUnderrun{singleShot, warning, 0x8C30};
IOLink_event measurementOverrun{singleShot, warning, 0x8C10};

//User task for reading sensor and updating deviceData Variable
void sensorTask() {
    int sharpDistance = distance();
    arduinoIOLink.updateDMSMeasurement(sharpDistance);
    if (sharpDistance < 10){
        arduinoIOLink.setEvent(&measurementUnderrun, 1);
        arduinoIOLink.setDeviceStatus(outOfSpec);
    }
    else if (sharpDistance > 80){
        arduinoIOLink.setEvent(&measurementOverrun, 1);
        arduinoIOLink.setDeviceStatus(outOfSpec);
    }
    else arduinoIOLink.setDeviceStatus(operating);
}

void setup() {
    arduinoIOLink.initDevice(sensorTask, deviceData); //Init IO-Link Device Settings
}

void loop() {
    arduinoIOLink.ioLink_Task();
}

int distance()
{
    int samples = 20;
    long sum = 0;
    for (int i = 0; i < samples; i++)
    {
        sum = sum + analogRead(A0);
    }
    float average = sum / samples;
    float distance_cm = 17569.7 * pow(average, -1.2062);
    return (int)distance_cm;
}
```

Figure 71 Arduino sketch for distance sensor

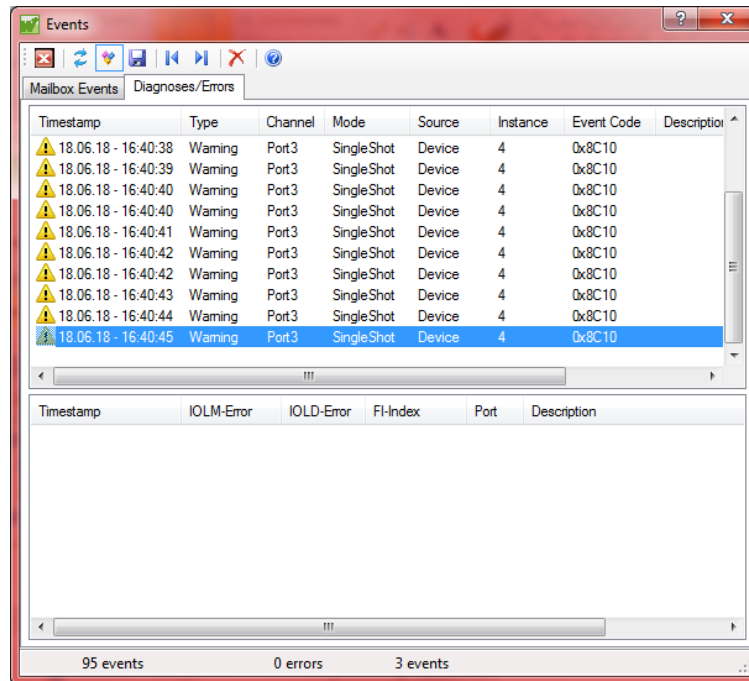


Figure 72 Test of IO-Link events triggered by the distance sensor

Gateway application

For this example, we use the PLC utility to read the parameters from the IO-Link device (see Figure 73). Afterward, the distance values from the sensor are read as seen in Figure 74 and then displayed in a GUI (see Figure 75). From the main program of the PLC, it can be seen that the size of input data received is the same as the one set in the GUI.

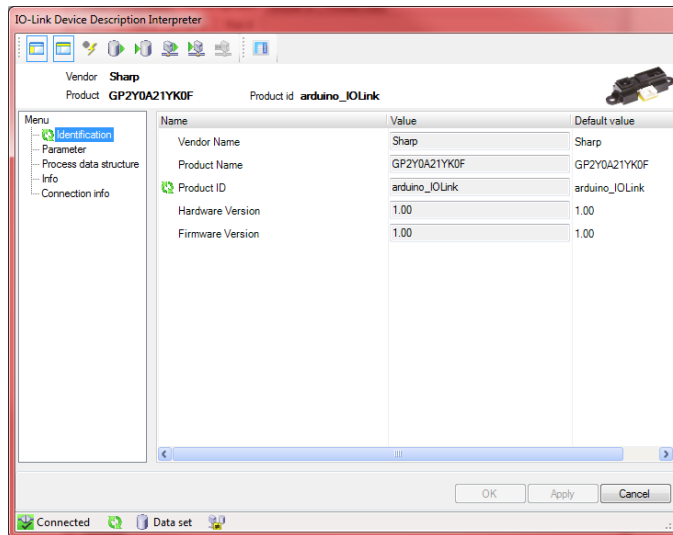


Figure 73 Reading IO-Link parameters of the distance sensor

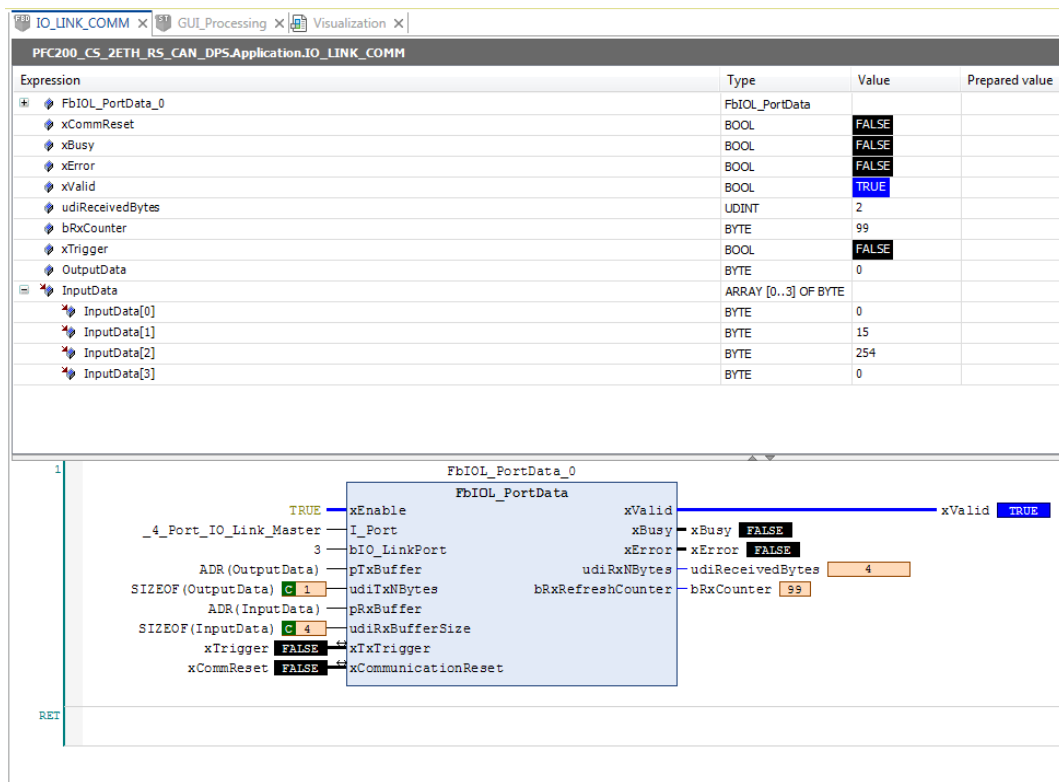


Figure 74 WAGO PLC program for the distance sensor

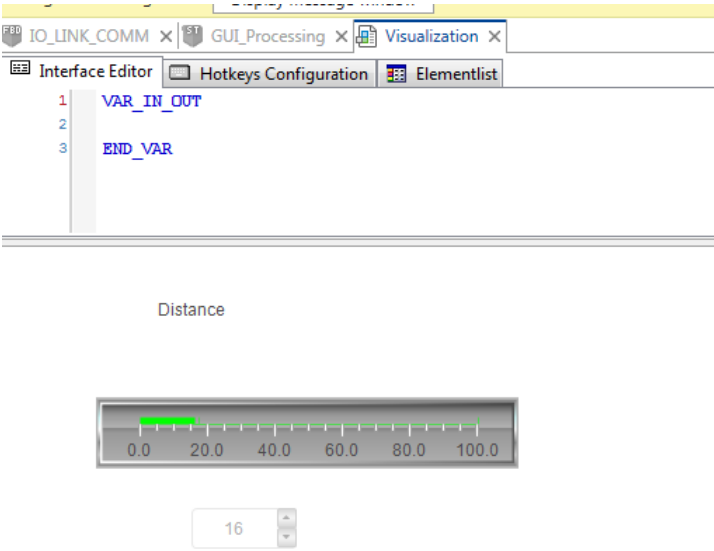


Figure 75 GUI for PLC program of the distance sensor

Conclusions

The creation of this framework allowed creating a development environment for low-cost IO-Link devices. Its implementation with example applications has demonstrated the simplicity of the workflow required for its operation. In addition, the end-user only needs to have a general overview of the IO-Link protocol without understanding the protocol's internal layers.

Furthermore, if the end-user needs to change any parameter of its IO-Link device it is only required to generate a new file configuration for the firmware and the IODD. Thus, it offers a versatile approach for the creation or modification of an IO-Link device on the fly.

Moreover, the firmware's API allows integrating any standalone sensor application seamlessly since it just requires adding a subroutine that will handle its sensor data in each IO-Link cycle.

The aforementioned advantages result in a real practical low-cost tool that can reduce development time of IO-Link sensors. Consequently, it contributes to an effortless incorporation of sensor applications to the latest set of integrating technologies known as Industry 4.0.

Future work

The present work has the possibility of several improvements. For instance, the framework capabilities were designed to bring a low-cost solution but could be expanded by porting the firmware to a microcontroller with more resources.

Likewise, the number of custom parameters the framework supports depends on the writing time of the built-in EEPROM of the microcontroller. The use of an external non-volatile memory with a faster response could solve this issue. Additionally, the available memory for the end-user could be increased if the firmware code is further optimized.

Finally, the GUI could be improved in two ways. The first one is adding an option so the IODD file could be created for other languages as described in [14]. The second improvement consists of porting the GUI software to other operating systems with the purpose of making it a cross-platform solution.

Bibliography

- [1] F. Mosconi, *The New European Industrial Policy: Global Competitiveness and the Manufacturing Renaissance*. London: Routledge, 2015.
- [2] A. Ustundag and E. Cevikcan, *Industry 4.0: Managing The Digital Transformation*. 2018.
- [3] IO-link Community, “IO-Link Interface and System Specification v1.1.2,” no. July. IO Link Community, Karlsruhe, 2013.
- [4] J. F. Wollert, “IO-Link for smart sensors,” 2015. [Online]. Available: <https://www.elektroniknet.de/elektronik/automation/fuer-smarte-sensoren-119458.html>. [Accessed: 23-Jul-2018].
- [5] IO-Link, “IO-Link System Description.” IO-Link Community, Karlsruhe, 2013.
- [6] IO-link Community, “IO-Link Common Profile v1.0,” no. March. IO-Link Community, Karlsruhe, 2017.
- [7] T. Instruments, “TIOL111-5 and TIOS101-5 Evaluation Modules.” 2017.
- [8] J. L. Maria Soto, Marc Sevaux, André Rossi, *Memory Allocation Problems in Embedded Systems*. Wiley-ISTE, 2012.
- [9] Atmel, “ATmega328 / P,” *AVR Microcontrollers*, p. 442, 2016.
- [10] Reema Thareja, *Data Structure Using C*, 2nd ed. New Delhi: Oxford University Press, 2014.
- [11] IO-Link Smart sensor profile group, “IO-Link Smart Sensor profile 2nd Edition.” IO Link Community, Karlsruhe, 2017.
- [12] Wilbert O. Galitz, *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, 3rd ed. Indiana: John Wiley & Sons, 2007.
- [13] Riverbank, “What is PyQt?” [Online]. Available: <https://riverbankcomputing.com/software/pyqt/intro>. [Accessed: 30-Apr-2018].
- [14] I.-L. Consortium, “IO-Link Device Description v1.1.” Karlsruhe, 2011.
- [15] “IO-Link Downloads,” 2018. [Online]. Available: <http://www.io-link.com/en/Download/Download.php>. [Accessed: 02-May-2018].

Annexure 1
Thesis Proposal Outline

Name of Student: Víctor Francisco Chávez Bermúdez Student Id: 3141277

1. Areas of interest and Justification of Research/Project

Areas of Interest

- Generic sensors
- Plug and play sensor interface for PLC
- Embedded systems
- Industry 4.0
- Communication protocols for sensors

Justification of Research/Project

Enabling generic sensors based on IO-Link technology
Easy access prototyping framework for IO-Link sensors
Contribution to IO-Link technology development.
Faster development of IO-Link based applications.

2. Research/Project Goals and Objectives

Research/Project Goals

- Design and development of an IO-Link framework for multipurpose generic sensors.
- Design and development of generator for generic IO-Link Device Description (IODD) files.
- Proof of concept by IO-Link sensors based on the framework

Research/Project Objectives

- Research about IO-Link protocol specification and related documents.
- Software design and development of an IO-Link stack device.
- Hardware design and development for generic IO-Link devices.
- Software design and development of an IODD utility.
- Deployment of a proof of concept for generic sensors based on IO-Link framework.

- Creation of user documentation for the proper use of the IO-Link framework software and hardware.

3. Summary of Preliminary Literature Review

IO-Link Interface and system specification v1.1.2
IO-Device description specification v1.1
IO-Link Common profile specification v1.0
IO-Link Smart Sensor Profile 2nd Edition V1.0
Wago IO-Link Master documentation
IFM Electronics IO-Link sensors documentation

4. Proposed institution, proposed technical advisor and methodology

Institution	FH Aachen
Name of Technical Advisor	Prof. Dr. Jörg Wollert

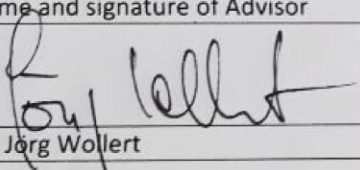
Methodology (brief description of how the goals and objectives are going to be achieved)

- Testing of different IO-Link device and master development kits as a learning tool.
- Reading and understanding of IO-Link specification.
- Software and hardware design for the IO-Link framework.
- Software design and development for a dynamic generator of IODD's.
- Creation of user documentation for the generic sensor framework based on IO-Link.

5. Anticipated Results

IO-Link framework software and hardware prototype
IODD utility software for the dynamic generation of IODD's
Hardware prototype for generic sensors with an IO-Link framework.
Proof of concept for generic sensors based on IO-Link framework.
User documentation for the generic sensor framework based on IO-Link.

6. Proposed Academic Advisor(s)

Name and signature of Advisor	Home institution:
	FH-Aachen
	Company associated:
	IFM Electronics
Dr. Jörg Wollert	Date: 06.04.18

Annexure 2
Zulassung zur Master-Abschlussarbeit

FH Aachen | Prüfungssekretariat FB8 | Goethestraße 1 | 52064 Aachen

Herrn
Victor Francisco Chávez-Bermúdez
Teresa Dasch
Theater str. 25
52062 Aachen

Zulassung zur Master-Abschlussarbeit

Sehr geehrter Herr Chávez-Bermúdez,

auf Ihren Antrag vom 11.06.2018 lasse ich Sie zur Abschlussarbeit zu. Das Thema und den Betreuer Ihrer Abschlussarbeit entnehmen Sie bitte der von Ihnen eingereichten Unterlage im Anhang.

Der Abgabetermin ist der 29.10.2018!

Der früheste Abgabetermin ist, unter Berücksichtigung des Postweges, der 17.09.2018!

Bis zu diesem Zeitpunkt legen Sie die Abschlussarbeit bitte in dreifacher Ausfertigung im Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik, Goethestraße 1, vor. Eine Kurzfassung in digitaler Form übergeben Sie bitte Ihrem Prüfer.

Ein Exemplar der vorliegenden Abschlussarbeit erhalten Sie sofort mit Siegel und Datum zurück. Dieses Exemplar halten Sie fünf Jahre für die FH Aachen vor, was Sie mit der Aufbewahrungspflicht dokumentieren.

Die Abschlussarbeit muss die schriftliche und unterschriebene Zusicherung enthalten, dass sie von Ihnen selbständig angefertigt wurde und keine anderen Quellen benutzt wurden, als die von Ihnen angegebenen.

Es sind folgende Prüfungsleistungen/ Unterlagen bis zum Kolloquium zu erbringen:

- **Sensors and Actuators 81623**
- **Entlastungsbescheinigung**

Für die Zulassung zum Kolloquium vereinbaren Sie nach Abgabe Ihrer Abschlussarbeit einen Termin mit Ihrem Prüfer.

Mit freundlichen Grüßen



Prof. Dr.-Ing. Manfred Enning

Anlage: Thema der Abschlussarbeit

FH Aachen
Goethestraße 1
52064 Aachen
www.fh-aachen.de

Prüfungsausschuss
Prof. Dr.-Ing. Manfred
Enning

Fachbereich 8
Maschinenbau und
Mechatronik

Auskunft erteilt
G. Zivkovic
T +49. 241. 6009 52321
F +49. 241. 6009 52681

pruefungsamt.maschbau@fh-aachen.de

Datum
12.06.2018

Einzureichen beim Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik

29.10
17.09.

Antrag auf Zulassung zur Abschlussarbeit

Bachelorarbeit Masterarbeit

Name und Vorname: Chavez Bermudez, Victor Francisco

Matrikelnummer: 3141277

Studiengang/ Studienrichtung: Mechatronik

Anschrift: Theaterstr. 25 z.Hd. Teresa Dasch

Geburtsdatum und -ort: 01.01.1994, Mexiko

Telefon und Mailadresse: 0163-789-3508 victor.fco.chavez@gmail.com

Betreuende/r Prof.: Prof. Dr.-Ing. Jörg Wollert

Korreferent (falls = externer Betreuer, s.U.): Prof. Dr. rer. nat. Klaus-Peter Kämper

Externer Betreuer (falls zutreffend; Titel, Name, Unternehmen bzw. Institution, Tel., Email):

Titel der Abschlussarbeit (max. 100 Zeichen inkl. Leerzeichen. Bitte in Normschrift oder maschinell ausfüllen; bitte beachten Sie: der Titel kann nach der Anmeldung nur auf Antrag innerhalb der ersten 4 Wochen geändert werden):

Design and development of a dynamic IO-Link based framework for generic sensors.

Ich erkläre durch meine Unterschrift, dass ich die Voraussetzungen für die Zulassung zur o.a. Abschlussarbeit erfülle und die geforderten Unterlagen dem Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik der FH Aachen bereits vorliegen oder dem Antrag beigefügt sind. Ich erkläre durch meine Unterschrift, dass ich bisher noch keinen Versuch zur Ablegung einer Abschlussarbeit unternommen habe. Mir ist bekannt, dass mit dem Datum zur Unterschrift meines Betreuers die Frist zur Abgabe der Abschlussarbeit beginnt!

11.06.18

Datum/Unterschrift der/des Studierenden

Ich erkläre mich bereit, die Abschlussarbeit zu betreuen.

11.06.18

Datum/Unterschrift der/des betreuenden Prof.

Wird die Abschlussarbeit im **Ausland** durchgeführt?

Ja Nein

Falls ja: Datum/Unterschrift der/des **Auslandsbeauftragten**

Antrag eingegangen:

11.06.18

Datum/Unterschrift **Prüfungssekretariat**

Sensors2 Act. 81623

Annexure 3
API for IO-Link firmware

The API for the firmware consists of methods that the end user may use to initialize the framework, get any status from the framework, set specific IO-Link configurations, and send process data through the protocol.

Some of the methods are only available if specific preprocessor directives are set in the configuration file of the framework. For the correct operation of the framework, the user must not use Timer 1 from the atmega328p since it will modify the behavior of the framework and will not function properly.

initDevice

Function	Initialization of IO-Link framework should be called only once e.g. setup function in Arduino framework
Prototype	<code>void initDevice(void (*userFunction)(), uint8_t *data)</code>
Parameter	<code>void (*userFunction)()</code> : Function that will be executed at the end of each cycleTime, where the user may use for a sampling of the sensor and other user-specific operations. <code>uint8_t *data</code> : User data that is meant to be sent to Master IO-Link i.e. Process input data. If process input data length equals zero, this parameter should not be used. Note: The userFunction execution time may not exceed the device cycleTime since it will block the device communication with the IO-Link Master
Return	none

begin

Function	Indicates if it's the first time the framework is going to be downloaded to the microcontroller. Must be used only the first time the program will be uploaded to the microcontroller. If this method is called each time the microcontroller starts the default parameters will always be loaded and the EEPROM values will be ignored.
Prototype	<code>void begin(void)</code>
Parameter	none
Return	none

addParameter

Function	Add a custom parameter to the IO-Link device. Should be called in the setup section of the program.
Prototype	<code>bool addParameter(uint8_t * parameterAddress, uint8_t length, bool writeAvailable)</code>
Parameter	<code>uint8_t * parameterAddress</code> : Address of parameter to be stored. Note: It should be of type <code>uint8_t</code> or <code>uint16_t</code> <code>uint8_t length</code> : Length of the parameter in bytes <code>bool writeAvailable</code> : Indicate whether the parameter is writeable, i.e. can be saved in EEPROM.

Return	<i>True</i> if success, <i>False</i> if the framework was not able to add the parameter (i.e. total number of that can be added is greater than the ones indicated in the header configuration file).
--------	---

ioLink_Task

Function	Executes all the state machines involved in the framework. It should be called once inside the main loop of the program.
Prototype	<code>void ioLink_Task(void)</code>
Parameter	none
Return	none

setEvent

Function	Notifies the IO-Link master of any event occurred in the device.
Prototype	<code>bool setEvent(struct IOLink_event * events, uint8_t numberEvents)</code>
Parameter	<p><code>struct IOLink_event * events</code>: Address in memory of structure of events that should be sent to IO-Link Master.</p> <p>Initialization of an event may look as follows:</p> <pre> IOLink_event myEvent{ singleShot, //IO-Link mode(singleShot,appears,disappears) notification, //IO-Link type (notification,warning,error) 0xFF91 //16 bit code as specified in Annex D of IO-Link spec. }; </pre> <p><code>uint8_t numberEvents</code>: number of event structures to be sent to IO-Link Master</p>
Return	<i>True</i> if events were able to be sent

	<i>False</i> if event can't be sent (i.e. if there are previous Events in the queue to be sent)
--	---

isPDOValid

Function	Checks whether the process output data from Master is valid
Prototype	bool isPDOValid (void)
Parameter	none
Return	<i>True</i> if Process output data is valid. <i>False</i> if Process output data is invalid.

getPDOOut

Function	Get the Process output data from the IO-Link master
Prototype	void getPDOOut (uint8_t *data)
Parameter	uint8_t *data: Address of variable where Process output data will be copied.
Return	none

IsCOMLost

Function	Checks if the communication with master has been lost
Prototype	bool isCOMLost (void)
Parameter	none
Return	<i>True</i> if Communication with Master has been interrupted <i>False</i> if Communication with Master has not been interrupted

controlPDIn

Function	Indicate to Master whether device data (process input data) is valid or invalid e.g. User may specify Invalidity of data when sensor data is corrupted.
Prototype	<code>void controlPDIn(bool validity)</code>
Parameter	bool <i>validity</i> : <i>True</i> if Process input data is Valid <i>False</i> if Process input data is Invalid
Return	none

setDeviceStatus

Function	If any Smart sensor profile is active, change the status of the device as indicated in the Interface Spec v1.1.2 Annex B.2.18
Prototype	<code>void setDeviceStatus(deviceStatusParameters mode)</code>
Parameter	<code>deviceStatusParameters mode</code> : Device status mode, permitted values are: <i>operating, maintenanceReq, outOfSpec, functionalCheck, failure</i>
Return	none

isSensorDisabled

Function	If Smart sensor profile FSS is configured in the framework, check if Master has requested to disable the sensor
Prototype	<code>bool isSensorDisabled(void)</code>
Parameter	none
Return	<i>True</i> if Master requests that the sensor of the device is disabled (e.g. Sleep or turn off sensor). <i>False</i> if Master hasn't requested to disable sensor.

setSSC

Function	If Smart sensor profile FSS is configured in the framework, set SSC signal.
Prototype	<code>void setSSC(uint64_t measurement, uint8_t vendorSpecByte=0)</code>
Parameter	<code>uint64_t measurement</code> : Measurement value from sensor. <code>uint8_t vendorSpecByte</code> : Vendor specific bytes set by the user application (max. 7 bits long).
Return	none

updateDMSMeasurement

Function	If Smart sensor profile DMS is configured in the framework, update DMS measurement from device sensor
Prototype	If high resolution disabled: <code>void updateDMSMeasurement(int16_t data, uint8_t vendorSpecByte=0)</code> If high resolution enabled: <code>void updateDMSMeasurement(int32_t data, uint8_t vendorSpecByte=0)</code>
Parameter	<code>int16_t/int32_t data</code> : Sensor measurement value can be either 16 or 32-bits long depending on if the device has high-resolution support. <code>uint8_t vendorSpecByte</code> : Vendor specific byte (Max. 8-bits long)
Return	none

Annexure 4
GUI user manual

This section describes the parameters the end-user may modify in the GUI, a description of its purpose and the values it may take.

General functions

The main window of the GUI contains the general functions that the end-user can use independent of the parameters of each of the tabs. Figure 76 shows the main functions of this window.

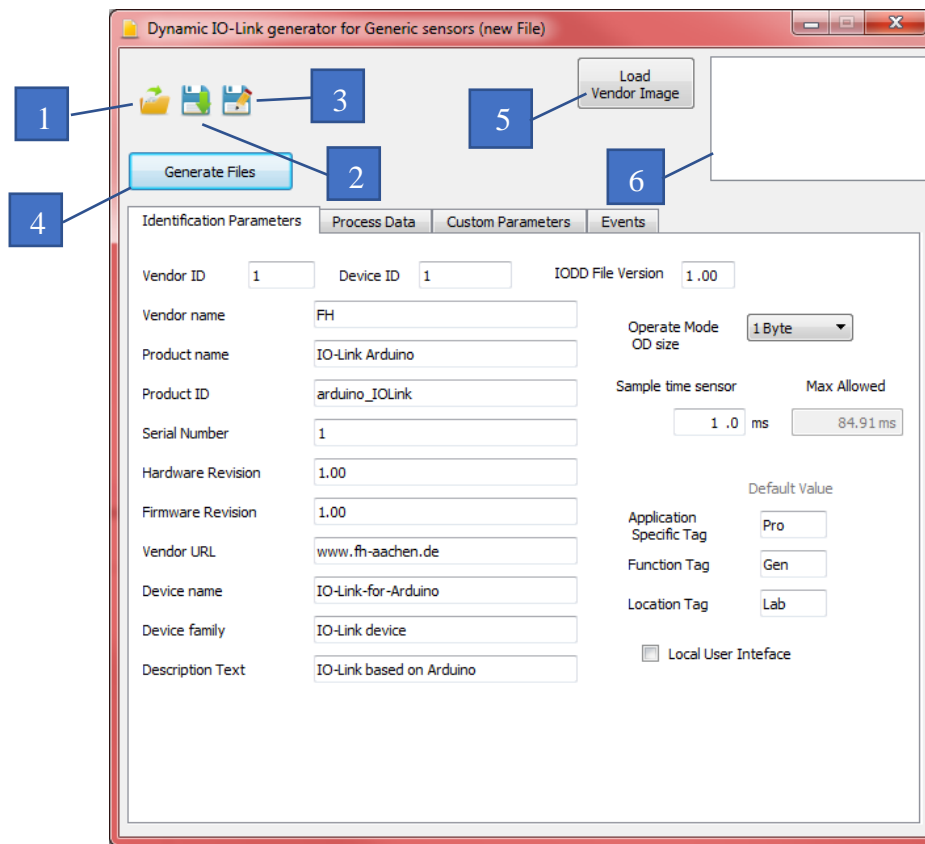


Figure 76 Main window of GUI

<i>Number</i>	<i>Function</i>	<i>Description</i>
1	<i>Open File (Ctrl + O)</i>	<i>Open previously saved configuration file.</i>
2	<i>Save File (Ctrl + S)</i>	<i>Save current configuration.</i>

3	<i>Save File As (Ctrl + Alt + S)</i>	<i>Save as current configuration.</i>
4	<i>Generate Files</i>	<i>Generate the IODD File, Configuration File for the firmware and a sketch example as an aid for the User. The output files are generated in the directory \Output and are subdivided in folders named after the vendor name.</i>
5	<i>Load Vendor Image</i>	<i>Load an image to be used as vendor logo for the IODD file. Accepted formats are JPG, PNG.</i>
6	<i>Vendor Image visualizer</i>	<i>Visualize a preview of the loaded image.</i>

Table 10 General functions of GUI

Identification parameters Tab

This tab contains the mandatory identification parameters that an IO-Link device needs in order to function properly (see Figure 77). If the Smart sensor profile mode is disabled (from Process data Tab) some of these parameters are not required. The parameters of this tab are described in Table 11.

The screenshot shows the 'Identification Parameters' tab selected. The interface includes the following elements:

- Navigation tabs: Identification Parameters (selected), Process Data, Custom Parameters, Events.
- Vendor ID: Device ID: IODD File Versi:
- Vendor name: Operate Mode: (dropdown)
- Product name: OD size: (dropdown)
- Product ID: Sample time sensor: ms Max Allowed: ms
- Serial Number:
- Hardware Revision:
- Firmware Revision:
- Vendor URL:
- Device name:
- Device family:
- Description Text:
- Application Specific Tag: (Default Value)
- Function Tag: (Default Value)
- Location Tag: (Default Value)
- Local User Interface

Figure 77 Identification parameters tab

<i>Option</i>	<i>Description</i>
Vendor ID	Unique value corresponding to a vendor. Value must be between the range of 1 to 65535.
Device ID	Identification of device from vendor. Value must be between the range of 1 to 16777215
IODD File Version	Contains the version of the concrete instance and not the version of the IODD specification. The vendor shall increase this version for each official release of the IODD for a particular device. Format must be of type “xx.xx” where x is a number between 0 and 9
Vendor name	Name of the vendor represented (max. 64 chars).
Product name	Name of the product represented (max. 64 chars).
Product ID	Vendor-specific product or type identification of the device (max 64 chars).
Serial number	Unique vendor specific code for each individual device (max 16 chars).
Hardware revision	Vendor-specific coding for the hardware revision of the device (max 64 chars).
Firmware revision	Vendor-specific coding for the firmware revision of the device (max 64 chars).
Vendor URL	Vendor’s URL (max 64 chars). Must have xsd:string with pattern: [A-Za-z][A-Za-z0-9 _-]*[A-Zaz0-9]
Device name	Common name for all variants. Must have xsd:string with pattern: [A-Za-z][A-Za-z0-9 _-]*[A-Zaz0-9]
Device Family	Vendor-specific classification of the devices Must have xsd:string with pattern: [A-Za-z][A-Za-z0-9 _-]*[A-Zaz0-9]
Description Text	Descriptive text of the device. Must have xsd:string with pattern:

	[A-Za-z][A-Za-z0-9 _-]*[A-Zaz0-9]
Operation mode OD Size	Number of On request data bytes to be transmitted from the device.
Sample time sensor	Time it takes for the specific user application to read sensor and execute any other operation between a cycle Time in milliseconds.
Max allowed	Displays max permitted value for “sample time sensor” depending on currently configured parameters in milliseconds.
<ul style="list-style-type: none"> • Application Specific Tag • Function Tag • Location Tag 	Optional tags for the device, e.g. to describe a specific application of IO-Link device (max. 3 chars each).
Local user interface	Select if the device has a local HMI which can modify device parameters.

Table 11 Parameter identification Tab Options

Process Data Tab

As shown in Figure 78, this tab contains information describing the process data that goes from the device (input) and to the device (output). If the user wants to create its custom process data structure, an “add” and “delete” button enables the user to do it dynamically for its application. The parameters the user may modify for process data are the data type, the name of the parameter and the size for process data. In addition, the number of bytes that the process data will occupy in total are displayed as additional information. Further information is described in table Figure 78

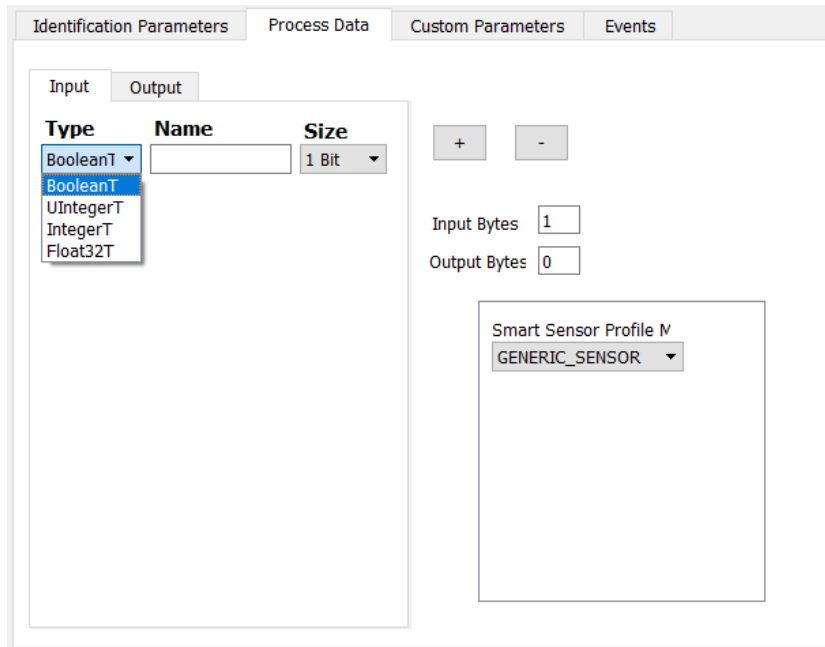


Figure 78 Process data Tab

Option	Description
Input/Output Tab	Process data structure for the corresponding tab. Each individual element has: <ul style="list-style-type: none"> • Type: Type of variable • Name: Descriptive name of the variable • Size: Size of variable
+ button	Adds a variable in the currently selected Process data tab (Input/output)
- button	Deletes a variable in the currently selected Process data tab (Input/output)
Input bytes	Displays total size of Process input data structure (max. allowed 32 bytes)
Output bytes	Displays total size of Process output data structure (max. allowed 32 bytes)
Smart Sensor profile Mode	Depending on the selected mode, it may modify the process data structure to meet the smart sensor profile mode as described in [11].

	<ul style="list-style-type: none">• GENERIC_SENSOR: Process data structure doesn't have a standard structure.• DMS: Digital measuring sensors, it's purpose is to standardize the data structures for measuring sensors.• FSS: Fixed switching sensors, these are Devices offering exactly one binary output signal (switching signal). The Setpoint of this switching output is predefined during the manufacturing process and is, therefore, fixed for the application.• DISABLE: No smart sensor profile is used, preferred for small simple applications.
--	---

Table 12 Process data Tab options

Custom parameters Tab

In the case, the user wants to add custom parameters this tab allows the user to add or delete parameters dynamically (see Figure 79). Further information is described in Table 13.

Identification Parameters Process Data Custom Parameters

Type	Name	Default Value	Write Access	
uint8_t	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	+
uint8_t				-
uint16_t				

Figure 79 Custom parameters input for the user in GUI

Option	Description
Type	Type of variable (uint8_t or uint16_t)
Name	Name of the variable (can't contain spaces and the first character can't be a number)
Default Value	Default value of variable depending on variable type. Uint8_t (max. value 255) Uint16_t (max. value 65535)
Write Access	If checked, the parameter will be saved in the data storage mechanism of the device (non-volatile memory).
+ button	Adds a custom parameter.
- button	Deletes custom parameter.

Table 13 Custom parameters tab options

Events Tab

The user may specify if the device uses standard or user-specific events as described in [3]. As shown in Figure 80, two internal tabs are available to the user to either select standard events or dynamically add custom events. Table 14 describes the available options in this tab.

	Standard	Custom
<input type="checkbox"/>	No malfunction	<input type="checkbox"/> Parameter error
<input type="checkbox"/>	General malfunction	<input type="checkbox"/> Parameter missing
<input type="checkbox"/>	Temperature fault - Overload	<input type="checkbox"/> Parameter changed
<input type="checkbox"/>	Device temp. over-run	<input type="checkbox"/> Wire break of a subordinate device
<input type="checkbox"/>	Device temperature under-run	<input type="checkbox"/> Short circuit
<input type="checkbox"/>	Device hardware fault	<input type="checkbox"/> Ground fault
<input type="checkbox"/>	Component malfunction	<input type="checkbox"/> Technology specific application fault
<input type="checkbox"/>	Non volatile memory loss	<input type="checkbox"/> Simulation active
<input type="checkbox"/>	Batteries low	<input type="checkbox"/> Process variable range over-run
<input type="checkbox"/>	General power supply fault	<input type="checkbox"/> Measurement range over-run
<input type="checkbox"/>	Fuse blown/open - Exchange fuse	<input type="checkbox"/> Process variable range under-run
<input type="checkbox"/>	Primary supply voltage over-run	<input type="checkbox"/> Maint. req. - Cleaning
<input type="checkbox"/>	Primary supply voltage under-run	<input type="checkbox"/> Maint. req. - Refill
<input type="checkbox"/>	Secondary supply voltage fault	<input type="checkbox"/> Maint. req. - Exchange wear and tear parts
<input type="checkbox"/>	Device software fault	

Figure 80 Events tab

Option	Description
Standard	Selection of any standard events the user device may use.
Custom	User-specific custom Events (see Figure 81)

Table 14 Events tab options

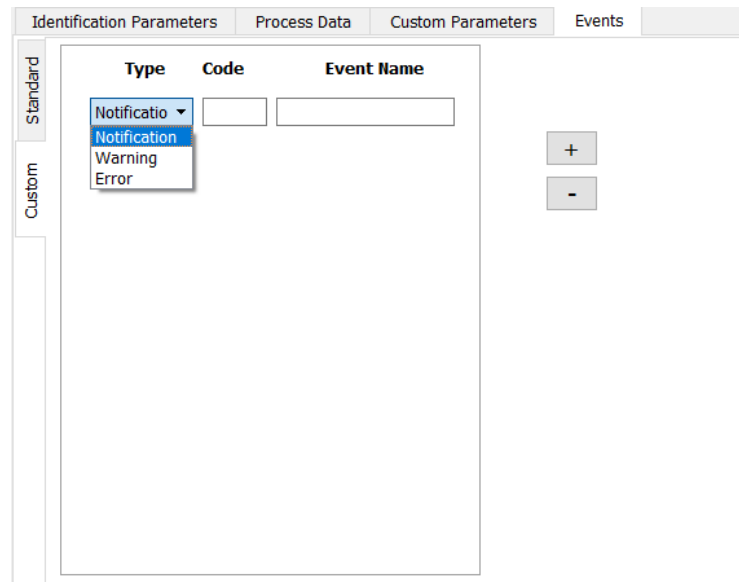


Figure 81 Custom events tab

Option	Description
Type	Type of Event (Notification, Warning, Error)
Code	16-bit event code in hex format e.g. 0x00 Valid range 0x1800 to 0x18FF or 0x8CA0 to 0x8DF
Event Name	Custom Event name
+ button	Add a custom event
- button	Delete a custom event

Table 15 Custom events tab options