u-create studio

System manual

Translation of the original instructions

Weidmüller

Document No.:
Filename: u_create_shen.pdf
Pages: 138

Weidmüller Interface GmbH & Co. KG Klingenbergstraße 26, 32758 Detmold, Germany, Phone: +49 5231 14-0, Fax: +49 5231 14-292083

Record of Revision

Version	Date	Change in chapter	Description	Changed by
00	06-2019	-	Newly created	hli, hasl
01	08-2019	Access to flash storage medium	Updated	hasl
1.02	04-2020	Various chapters	Updated to new version	hasl
1.03	06-2020	Licenzing, Modbus server	Updated change device, new chapter	hasl
1.04	08-2021	Software Units, Various chapters	Chapter deleted, updated to new version	hasl
1.05	11-2021	Various chapters	Hotifx, firmware update, DevAdmin updated	hasl

Table of contents

1 Introduction

1.1 Purpose of the document

This document describes the structure of u-control.

In addition, it describes the assembly and installation, the wiring aswell as the operation and displays of the modules.

The installation and configuration is described enough to obtain an operationally ready system. "Ready for operation" means that the system and/or the respective CPU modules are ready for loading the customer application.

Information

This manual is not addressed to end customers! Necessary safety notes for the end customer have to be taken into the customer manual in the respective national language by the mechanical engineers and system providers.

1.2 Requirements

The system manual is written for everyone using a u-create studio system in comination with a UC20-SL2000-EC or UC20-SL2000-EC-CAN or who plan to use such a system.

Only electricians trained pursuant to VDE 1000-10 are allowed to install and service a system with the help of this system manual.

Target group	Requisite knowledge and abilities
Safety engineer for "Functional Safety"	Basic technical training (technical college, engineer training or corresponding professional experience). Knowledge of: ● Principles of functional safety ● All current standards and safety regulations for the machine/ plant, in particular also knowledge of validation conforming to EN ISO 13849-2. ● Special risk potential of the machine/system and the production process ● Specific protective measures to avert machine-specific hazards (based on hazard and risk analysis) ● In-depth knowledge of national accident prevention regulations.
Project engineer	Technical basic education (advanced technical education, engineering degree or corresponding professional experience), Knowledge of: ● The method of operation of a PLC, ● Current valid safety information, ● The application.

Target group	Requisite knowledge and abilities
Electrician	Technical training in electrical engineering (based on industry-standard training guidelines). Knowledge of: ● Current valid safety information, ● Wiring guidelines ● Circuit diagrams ● System analysis and troubleshooting ● Professional manufacture of electrical connections in conformity with national and international regulations.
Programmer	Technical training (University of Applied Science, engineering degree or corresponding professional experience). Knowledge of: ● The method of operation of a PLC, ● Current valid safety information, ● Programming a PLC (IEC61131).
Start-up operator	Technical basic education (advanced technical education, engineering degree or corresponding professional experience), Knowledge of: ● Current valid safety information, ● Method of operation of the machine or system ● Fundamental functions of the application ● System analysis and troubleshooting ● Setting options on the operating devices
Service technician	Technical basic education (advanced technical education, engineering degree or corresponding professional experience), Knowledge of: ● The method of operation of a PLC, ● Current valid safety information, ● Method of operation of the machine or system ● Diagnostic options ● Systematic error analysis and troubleshooting

1.3 Intended use

u-control may only be used for the types of use described in the technical descriptions and only in conjunction with recommended/approved third-party equipment/installations.

All modules of u-control have been developed, manufactured, tested and documented in accordance with the appropriate safety standards. Therefore, the products do not pose any danger to the health of persons or a risk of damage to other property or equipment under normal circumstances, provided that the instructions and safety precautions relating to the intended use are properly observed.

The system is intended for installation in a control cabinet and may only be used in the following way:

- Only in the industrial sector
- in accordance with regulations
- In a faultless technical condition
- In its original state without unauthorized modifications.

Only alterations and modifications described in the accompanying documentation are authorized.

1.4 Notes on this document

Information

If necessary, also adhere to the documentation accompanying the modules.

1.4.1 Contents of the document

- System overview
- Explanation of all interfaces and displays of the devices
- General instructions for installation
- Operating behavior
- Description of the software installation
- Description of software interfaces and services
- Diagnosis and maintenance
- Tutorials

1.4.2 Not contained in this document

- Programming instruction
- Application diagnosis
- Firmware description

1.5 Further documentation

When using u-remote I/O modules, please also refer to the Remote I/O System u-remote manual. The documentation of the tools and libraries is integrated as online help.

1.6 Relevant countries and registrations

Standards and test values which the product conforms to and meets are contained in the chapters "Technical Data" and "EC directives and standards". For the EU guidelines relevant for product conformity please see the declaration of conformity.

2 Safety instructions

2.1 Representation

At various points in this manual, you will see notes and precautionary warnings regarding possible hazards. The symbols used have the following meaning:

DANGER!

indicates an imminently hazardous situation, which will result in death or serious bodily injury if the corresponding precautions are not taken.

WARNING!

indicates a potentially hazardous situation, which can result in death or serious bodily injury if the corresponding precautions are not taken.

CAUTION!

means that if the corresponding safety measures are not taken, a potentially hazardous situation can occur that may result in slight bodily injury.

Caution

means that damage to property can occur if the corresponding safety measures are not taken.

ESD

This symbol reminds you of the possible consequences of touching electrostatically sensitive components.

Safety information

Describes important safety-related requirements or informs about essential safety-related correlations.

Information

Identifies practical tips and useful information. No information that warns about potentially dangerous or harmful functions is contained.

2.2 General safety information

The documentation contains the information required to plan the use of u-create studio with UC20-SL2000-EC or UC20-SL2000-EC-CAN.

Familiarity with and correct application of the information contained in these manuals is a prerequisite for successful planning and safe installation, commissioning and maintenance of automation systems. Only qualified persons have the required technical expertise for correct interpretation and implementation of the instructions in these documents.

For reasons of clarity, not every single detail of every version of the products described is listed nor can every practical situation be taken into consideration. If you need further information, please request this from Weidmüller.

When installing, commissioning and servicing u-create products, the instructions given in the relevant part of the documentation must be observed.

2.3 Personnel safety

WARNING!

Unqualified interventions in the control may cause abnormal behavior of the machine/plant or personal injury or damage to the equipment. Only specially qualified staff may carry out activities on the control system.

2.4 Safety instructions for projecting

CAUTION!

- The instructions contained in the further delivered must be precisely followed in all circumstances. Failure to do so could result in the creation of potential sources of danger or the disabling of protective circuits integrated in the u-create system.
- Apart from the safety instructions given in these manuals, safety precautions and accident prevention measures appropriate to the situation in question must also be observed.
- Measures must be taken to ensure that in the event of power dips or power failures, an interrupted program can be properly restarted. In such situations, dangerous operating conditions must not be allowed to occur even temporarily.
- In all situations where faults occurring on the automation system could cause personal injury or significant damage to machinery and equipment, additional external safety measures must be taken in order to ensure the system as a whole remains in a safe operating condition even in the event of a fault.

3 System overview

The field of application of u-control is general machine and plant automation. The system features a modular design and can be set up according to the respective functional requirements.

It comprises the following components:

- Construction kit with I/O modules (u-remote) for various applications that can be added on.
- Toolsuite for automation tasks, consisting of tools for configuration, programming, visualization as well as for commissioning and diagnostics. Software libraries and interfaces are made available to provide support in the programming processes.

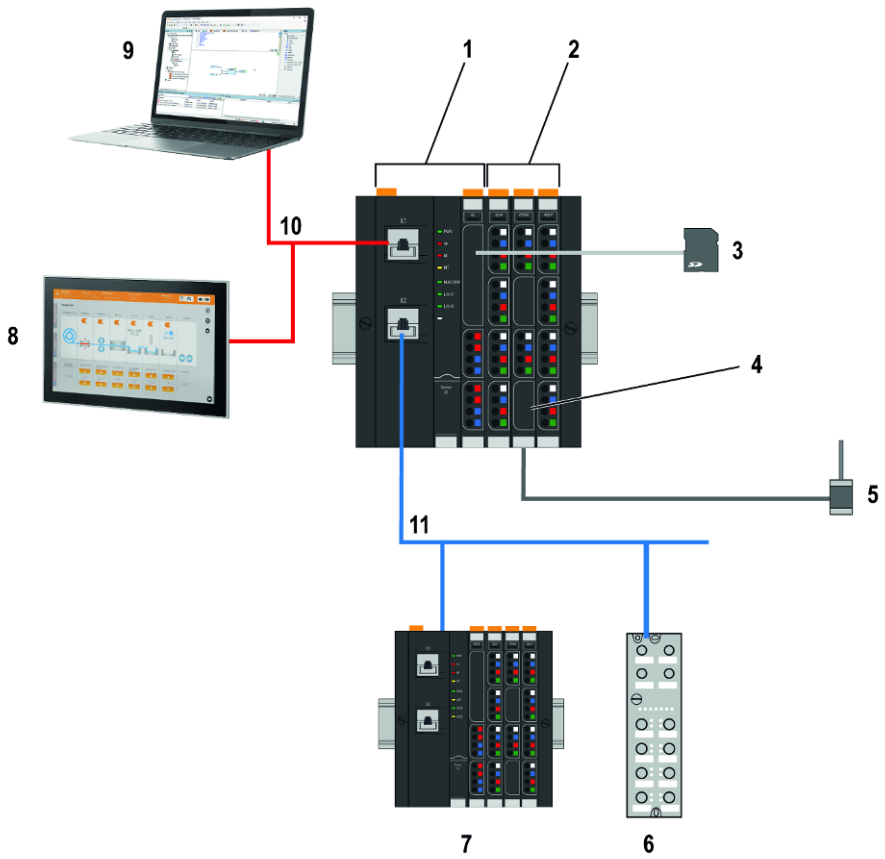


Fig. 3-1: Example system structure

1 ... Control	2 ... UR20 modules
3 ... SD card	4 ... PWM module
5 ... DC motor	6 ... UR67 module
7 ... UR20 station	8 ... Touch panel
9 ... Tool-Suite	10 ... Ethernet
11 ... Field bus	

Information

The system components shown in this manual are example graphics. The devices used by you may differ in their appearance. Please observe the supplied operation manuals of the devices.

3.1 Hardware architecture

Devices are connected according to their signal type, e.g. via analog or digital input or output modules, interface modules, etc.

CPU modules and I/O modules must be integrated into a control cabinet. Their enclosure only provides mechanical protection; the EMC shielding happens inside the device. For greater distances between the transducers a decentralized groups of I/O module clusters can be used. These can be connected with the CPU module via bus link modules.

The operating and display devices can be arranged at a suitable location somewhere on the machine/plant.

During the start of the system, the runtime system compares the current hardware configuration (actual configuration) with the hardware configuration (set configuration saved in the u-create studio project. Deviations in configuration or faulty modules can be identified via the inquiry of the module status in the IEC application. The response is set here via the program (e.g. error output on the visualization, restricted function in the optional I/O modules, etc.)

3.2 Software structure

The following graphic shows the structure of the software on the u-create system.

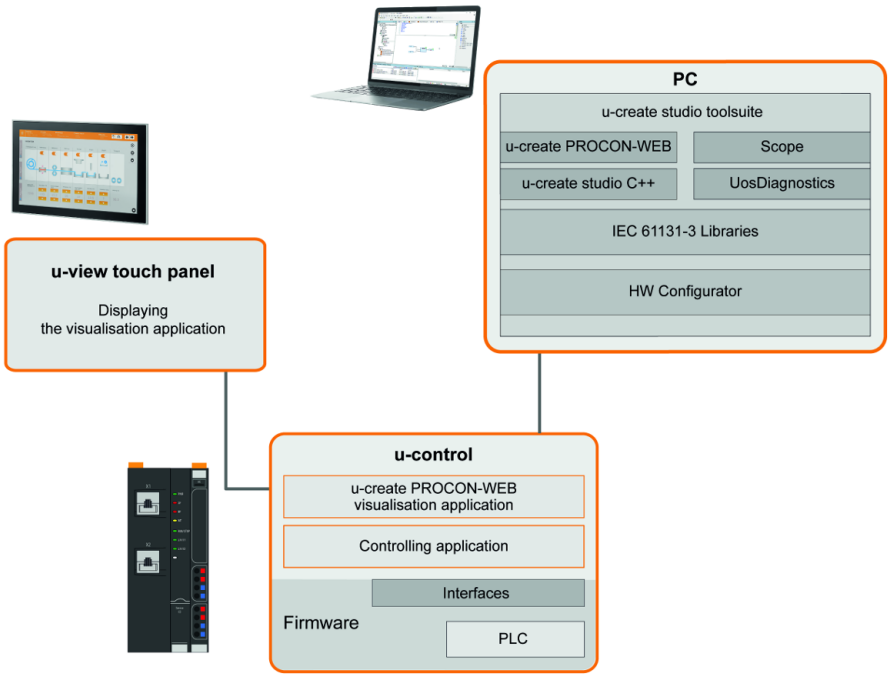


Fig. 3-2: Software overview

The control application runs on the control. There are also libraries and interfaces available via which the control can be accessed via the program.

The tool suite is installed on a PC. The programs are used for parameterization, programming, and diagnostics.

The control software is based on a Debian (Linux) operating system with customized scope.

In addition to the UNIX-based system, the KeBian system consists of a multitude of packages and is therefore individually configurable. In addition to a multitude of publicly available standard packages you can also create your own packages and integrate those in your system. A detailed description of the available standard packages can be found in the Debian documentation under http://www.debian.org/doc/. Under http://www.debian.org/distrib/packages you can find an official listing of packages and their package sources.

The following runtime licences support creating own packages:

- U-CREATE-STUDIO-RT-OLAC
- U-CREATE-STUDIO-RT-OLC

Information

The version designation of your u-create system can be found in the release notes supplied with the version!

The runtime system used on the control conforms to the IEC Standard in compliance with 61131-3. The following topics are covered:

- IEC programming: All languages

- C programming

3.3 Network design

The IP address of the network is typically specified by a network administrator. At initial startup of the control with u-create studio the IP addresses of the devices are not attuned with each other. The network settings of the PC or the controls therefore must be adjusted.

The communication connection between PC and control can either take place through a direct connection via an Ethernet cable between the Ethernet interface of the PC and the On-board-Ethernet of the control or via a switch.

If a DHCP server exists on the network, the network address can be obtained automatically. If no DHCP server exists, or if there is a direct connection between the control and the PC, a static network address must be used.

Information

The control and tools do not support NAT (Network Address Translation). This limitation must be taken into consideration, especially when operating via a VPN tunnel or through a firewall!

If a firewall is used in the system, port 1744 must be open for UDP traffic (incoming as well as outgoing). Otherwise a connection to the control will be impossible.

Please contact your network administrator for questions concerning the network operation.

The PC in the network can only communicate with the control. Communication with additional components of the system is not possible in this network structure.

The IP addresses can be defined when creating a target system ("Create Target"). The following values are preset by default.

Name	Address
Control (X1):	192.168.101.100
Subnet mask:	255.255.255.0
Gateway:	192.168.101.x
u-view Panel:	192.168.101.x
USB interface:	192.168.227.1

Information

If more panels are used different IP addresses must be set for them.

Information

To change the IP address of a panel the device must be connected to a network.

The IP addresses of the I/O modules are set in u-create studio EoE settings (see online help).

Information

A maximum of 3 handheld terminals can be used simultaneously.

Connection service PC - system components

For a service PC to be able to communicate with additional components, it must also be located in the machine network. For that, the service PC is either connected via a switch or directly to the X1 port of the control and must get an IP address by the user that is compatible with the network.

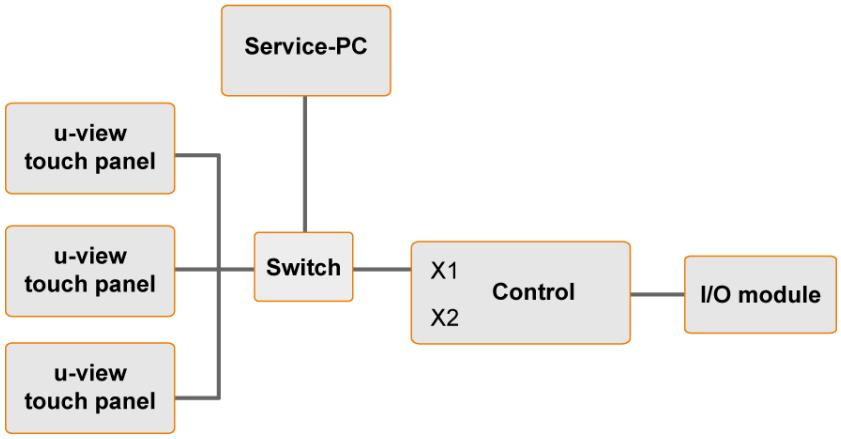


Fig. 3-3: Service PC in the machine network

3.4 u-control - CPU modules

Designation	Performance data	Interfaces
UC20-SL2000-EC	Dual Core A9, 512 MB RAM, 8 GB EMMC	• u-remote bus • EtherCAT Master
UC20-SL2000-EC-CAN	Dual Core A9, 512 MB RAM, 8 GB EMMC	• u-remote bus • EtherCAT Master • CANopen interface

For further information, please refer to the u-control manual or the product catalog (Automation section).

3.5 Buses

The sections below list the buses supported by the u-create system and describe them in more detail.

The following master fieldbus interfaces are available depending on the control version:

- EtherCAT
- CANopen
- Modbus TCP/IP

The following slave fieldbus interfaces are available depending on the control version:

- Modbus TCP/IP
- CANopen

3.5.1 EtherCAT

EtherCAT interfaces can be used for the connection of slave devices (e.g. drives, I/O modules).

Information

The X2 interface can also optionally be used as an Ethernet interface. Make sure that a bootloader of version 1.13 or higher is installed on the UC20-SL2000. You can upgrade via the firmware upgrade tool. If you have any questions, please contact support.automation@weidmueller.com.

3.5.2 CAN

When using the CAN interface, make sure to use UC20-SL2000-EC-CAN with CAN controller.

3.5.3 Modbus TCP/IP

Modbus is an open communication protocol for data exchange between a master, which requests or writes data, and slaves, which provide or receive data.

The Modbus protocol is specified for serial connections (Modbus RTU) and for Ethernet (Modbus TCP). Typical applications are transmission of values for non time-critical tasks. The data is provided in single register (16 bit values) or coils (discrete 1 bit values) via a Modbus table defined by the slave manufacturer.

Configuration

To communicate with a Modbus server on the control, configuration entries must be set via u-create studio. For detailed information see u-create studio online help.

3.6 u-create studio Toolsuite

The tools are all connected to the control via Ethernet. They are used for configuration, parameterization, programming, debugging and diagnostics on the control. The following tools are made available in the process:

- **Configuration and programming with u-create studio:**

  The programming environment u-create studio is used to configure, parameterize, program and debug control applications.

- **Creating C/C++ programs with u-create studio C++:**

  u-create studio C++ for creating and diagnosis/debugging of a control application in the programming languages C and C++.

- **Diagnostics usingu-create studio Scope:**

  u-create studio Scope is used to monitor, record and visualize the values of any variables. Unlike the debugger it will not require the program sequence to be interrupted.

The following chapters describe the tools in more detail.

3.6.1 u-create studio

u-create studio development environment offers a comfortable user interface with the following functions:

- Configuration and parameterization of the u-create system (including the PLC configuration)
- Programming according to EN 61131-3
- Integrated module libraries (see library descriptions)
- Library administrator for integrating additional libraries
- Debugging functions (testing program sequence, monitoring and modifying variables, error search).

Programming languages

u-create studio offers all 5 programming languages standardized in EN 61131-3. Each of these languages has specific characteristics that are ideally suited for carrying out specific tasks. Furthermore u-create studio offers the extension "Function block diagram (FBD)".

Programming language	Type	Description
Structured text (ST)	Textual programming language	"Structured text" comes closest to the programming languages Pascal and C used for the PC. It consists of a series of instructions that can be executed in high-level language ("IF..THEN..ELSE") or in loops (WHILE..DO).

Programming language	Type	Description
Function block diagram (FBD)	Graphic programming languages	Allow programming with logical symbols. It is very suitable for logic controls.
Continous graphic Function Chart (CFC)		In addition, based on the function chart there is the continuous graphic function chart (CFC) in which elements can be placed freely and feedback can be inserted directly.
Ladder Diagram (KOP)		The ladder diagram was developed from the circuit diagram. The representation of a ladder diagram program resembles a circuit diagram - relative to the representation of the logical links.
Sequential Function Chart (SFC)		The "Sequential Function Chart" is a chain of control steps that are connected by step-on conditions (transitions).

Library manager

To facilitate programming u-create studio makes it possible to organize objects that are not related to projects into libraries, such as modules, declarations and visualizations. For this purpose a library administrator is available for integrating and viewing of libraries.

3.6.2 u-create studio C++

u-create studio C++ allows to develop and integrate C or C++ programs in the Weidmüller automation system.

u-create studio C++ also offers the possibility to debug the created C or C++ programs already being processed on the control. Therefore a debug server is running on the control, the user interface for debugging is located in u-create studio C++.

Additionally the u-create system provides the following opportunities for C programming:

- **Cyclic C functions**

  Allows the integration of C functions. Thereby it is possible to integrate a function each at start-up, shut-down or when starting or stopping the system. For cyclical processing one or more tasks can be configured (with priorities, watchdogs, ...). For the access to the services of the control and the I/O system libraries which are already embedded into the u-create studio C++ are available.
  In u-create studio C++ the project type "Extended C Runtime Project"

serves as example. In this example a cyclical callback function and the event callback functions for state transitions of the control and its configuration are displayed.

- **Fast Control in C**

  Similar to the cyclical C function a C function can be integrated. This function is executed at a defined time indeed. Allows fast reading, processing and writing of values within one cycle. Function calls at Init/Start/Stop/Exit of the control can also be integrated.
  In u-create studio C++ the project type "Fast Control C Project" serves as example. In this example the cyclical callback functions for fast control and the event callback functions for state transition of the control and its configuration are displayed.

- **Independent C runtime system**

  Allows the integration of a C runtime system into the system which runs independently and can be integrated into the control procedures via Device Service. All tasks must be programmed completely. There is total access to POSIX/Linux and all interfaces provided by Weidmüller.
  In u-create studio C++ the project type "Custom Realtime Runtime System" serves as example. In this example the structural organisation of an independently running C runtime system and the interaction with the control procedure are displayed. Other examples with different functions are: "Executable Communication with the Controller" and "TCP Server Application".

- **Integration of C functions in IEC**

  Allows calling C function in the IEC. The cyclical processing happens within the context of the IEC program.
  In u-create studio C++ the project type "CoDeSys IEC Functions in C" serves as example. In this example the C part which can be called in the IEC is shown.

3.6.3 u-create studio Scope

u-create studioScope serves to monitor, record and visualize the values of arbitrary variables in software components. These can be firmware components or application programs (TeachTalk- or IEC-applications). Unlike the debugger, it does not require the program sequence to be interrupted. Tables and diagrams enable visualization of the recorded variable values.

The tool has the following properties:

- Variables and arrays in firmware components and teachtalk- and/or IECprograms can be recorded. The variables must be registered with the sampler.
- Variables can be activated or deactivated from scope. Only activated variables are recorded.
- Events can be used to notify a specific status in the target program.
- Related events can be grouped into classes. Every event is part of a class.

- The actual recording start can be determined via a trigger. The trigger can refer to a variables' value and/or a specific event. Pre- as well as posttriggering is possible.
- Representation of the collected data in different diagrams and views: variable monitor, event monitor, X-t chart, X-Y chart.

Information

Program execution may be slightly delayed while recording with data with u-create studio Scope.

For further information see the online help of u-create studio Scope.

3.7 Libraries and interfaces

Given below is a list of the software interfaces provided by the system.

3.7.1 IEC standard libraries

The following standard libraries are available in u-create studio:

- IEC61131-3 standard libraries
- Access to system functions
- Access to message system
- Diagnosis functions

For further information see the online help of u-create studio.

3.7.2 u-create studio libraries

With the specific libraries the following functions are available amongst other:

- Function blocks for accessing the message system and controller variables
- Function blocks for bus communication
- Function blocks for diagnosis

For further information see the online help of u-create studio.

3.7.3 OPC UA interface

The u-create system offers the opportunity to access to IEC variables of the control via an OPC UA interface. Therefore an OPC UA client which must be installed on a PC is needed. OPC UA clients are offered for download in the internet.

To establish a connection with the OPC UA client to the OPC UA server on the control the IP address or the hostname of the control with the port number `4842` first have to be input into the client. If a server has been found the endpoint "None - None" must be selected. Then the connection to the server

can be established and the data available on the OPC UA server (variables, functions, ...) are shown. For a detailled description see the help of the OPC UA client.

The server/endpoint URLs delivered of the OPC UA server always contain the hostname. For this reason it must be possible for the OPC UA client to solve the hostname. Some OPC UA clients do not offer his functionality. If this is not the case the control must be inserted manually into the file `C:\Windows\System32\drivers\etc\hosts` on the PC. Therefore a new line with the IP address and the hostname must be added in this file (e.g. `10.150.48.40  <Hostname of control>`).

3.7.4 REST-API (Representational state transfer)

The u-create system offers the possibility of a visualization connection via REST-API (Representational state transfer) interface. The interface description is available as OpenAPI specification in YAML format.

The API can be accessed via an Internet browser at the following address:

`<Control IP address>/api`

The current version of the REST API is displayed via this address. By appending the version to the address, the supported interface is displayed (e.g. `192.168.1.1/api/v3`)

The following services are offered and listed as interface file:

- `configuration.yaml`: Specifing basic configurations (e.g. network access, date and time, ...)
- `device-service.yaml`: Executing basic commands of the control (e.g. start, stop, ...)
- `diagnostics.yaml`: Supporting of diagnosis information (e.g. statusreport, ...)
- `license-text.yaml`: Reading "Open Source" license texts of the control
- `licensing.yaml`: Performing licensing (e.g. activating licenses, license overview, ...)
- `report-system.yaml`: Connection to the report system (e.g. reading out error, acknowledge error, ...)
- `update-service.yaml`: Performing a backup or restoring this
- `variables.yaml`: Reading and writing of released variables

By appending the respective file name to the Url, the functions and a description can be read out (e.g. `192.68.1.1/api/v3/configuration/configuration.yaml`)

4 Operating behavior

4.1 Start-up

The start-up of the PLC is divided into the following main stages:

- Start-up boot system
- Start-up control firmware
- Start-up application

The booting of the control starts automatically as soon as the CPU module is supplied with power. Conditions for the booting are:

- Voltage supply was connected correctly.

5 Software installation

For better scalability and to spare memory servers, the u-create studio has been divided into the following setups:

- Deliverable_Engineering_Environment: Setup for installation of the development environments
  - Setup_SimulationService (optional): Setup for installation of the control simulation (Simulation service, VBox)
- Deliverable_Server_Setup: Setup for installation of the device software
- Deliverable_FirmwareUpdate: Tool for performing a firmware update of the controller

Information

If a component has already been installed, the automatic installation does not support a new installation of this component. In this case, the respective component must be uninstalled using the individual product setup or via the operating system.

System requirements: Standard PC, Windows 10®

The setup "Deliverable_Engineering_Environment" is the installation of the development environments. The setup "Deliverable_Server_Setup" deals with server components that must be accessed by each development environment. Therefore there are the following possibilities of an installation:

- Single-user installation
- Network installation (recommended installation)

Single-user installation

With the single-user installation, all setups are executed on the PC. The development environments and the server components are installed on the same PC.

Network installation

With the network installation, the development environments are installed on the respective PC. The server components are installed once on a server in the network and are so available to all development environments. This installation offers the advantage of significantly reducing the installation time and memory requirements when installing the development environments on the PC. Multiple installations of the same server components (in a network) are also avoided.

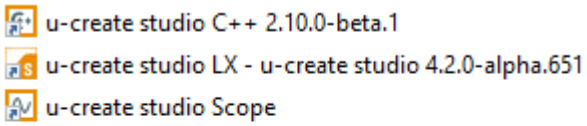
Installation "Deliverable_Engineering_Environment"

This setup installs the required development environments on the PC. Proceed as follows:

1) Open the installation directory.
2) Open the sub directory "Setup".
3) Execute application "setup.exe". The installation dialog opens.

4) Follow the instructions in the installation dialog.
5) By pressing "Install", the installation process starts.
6) By pressing "Finish" the installation ends.

All components are installed. The desktop now contains a link to a directory in which all necessary components can be started.



Information

All running applications must be closed during installation process.

Information

If the installation fails, temporarily disable existing virus scanners. Discuss the temporary uninstallation with your IT department.

Optional: Installation "Setup_SimulationService"

This setup installs the optional simulation service (control simulation) on the PC.

Information

If there is already an installed version of the software "VBox" on the PC, which is older than the version to be installed, it will be completely uninstalled and replaced by the supplied version. The required version can be taken from the release notes supplied.

Proceed as follows:

1) Open the installation directory.
2) Open the sub directory "Setup".
3) Execute application "Setup_SimulationService.exe". The installation dialog opens.
4) Follow the instructions in the installation dialog.
5) By pressing "Install", the installation process starts.
6) By pressing "finish" the installation ends.

The service "SimulationService" has been installed. The simulation can now be used.

Installation "Deliverable_Server_Setup"

This setup installs the required server components. Proceed as follows:

1) Open the installation directory.
2) Execute application "setup.exe". The installation dialog opens.
3) Follow the instructions in the installation dialog.
4) By pressing "Install", the installation process starts.
5) By pressing "finish" the installation ends.

All components are installed.

If the components have been installed on a server in the network (network installation), the access to them must still be configured in u-create studio. To do this, proceed as follows:

1) Start u-create studio
2) Open "Tools - Options"
3) Select "Software Service"

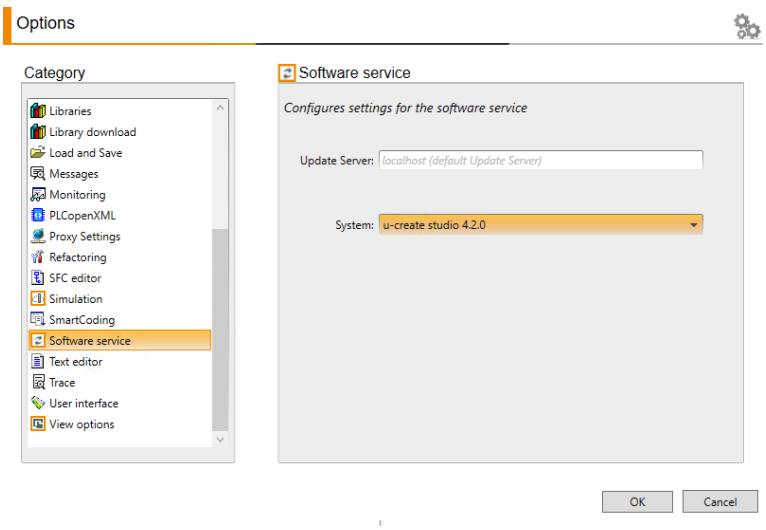


Fig. 5-4: Software Service

4) Setting the IP address of the server and the desired software version.

The configuration is done.

5.1 Install firmware of the device

Apart from the hardware-related software, no firmware is installed on some devices in the delivery state. This firmware (without hardware-related software) must be loaded onto a storage medium using u-create studio and installed on the device with this.

This requires an empty storage medium (FAT32 formatted) with a memory of at least 2 GB (max. 32 GB), which is plugged into the PC.

For further information see online help of u-create studio in chapter "Load firmware to device "

6 Configuration

The combined switching of the assemblies and modules in u-create studio that took place during the hardware installation is re-created with the configuration..

The control configuration is based on the device descriptions. These describe the basic configuration specifying which parameterization options are available. The control configuration allows the connection of I/O modules as well as fieldbus modules. Additional elements are available depending on the installed device descriptions.

The available I/O channels are indicated in the u-create studio control configuration and can be utilized in the application.

Information

Additional information as well as the exact parameter description of individual modules can be found in the online help of u-create studio.

The network configuration of the control is also possible vie Service App "DevAdmin" (see 10 Device Administration (DevAdmin)).

6.1 Data exchange between control and visualization

IEC system variables that have been added to a symbol file can be used by a visualization to display process values and allow operators to modify application variables.

u-create studio makes the so-called symbol file available for the exchange of system variables. The symbol file contains the IEC variables of the u-create studio project.

For detailed information about configuration or data exchange, please see the respective online help (u-create studio).

7 Program development

The control operates based on the following principle: "Read inputs, processing, right outputs". u-control carries out the following tasks during the processing of each task defined with u-create studio:

- Read inputs: At the beginning of the cycle, the current statuses of the inputs are read and written into the process map of the inputs.
- Processing: The application program is executed. The set status of the outputs is copied into the process map.
- Write outputs: At the end of the cycle, the set statuses of the outputs saved in the process map are transmitted to the physical outputs.

Furthermore it is possible to create several applications in parallel or nested and to start them on the control. A variable exchange between the applications is possible. For each application a symbol configuration can be created.

Details about the programming can be found in the online help of u-create studio.

7.1 Task priorities

IEC process priorities for a task can be assigned during programming. By default priority 10 is configured for a task.

The following tables describe the assignment of the IEC task to the mapped linux system task.

Reserved realtime tasks

Priority linux	Description
99 - 82	Tasks of this priority are reserved for system

High prior real time tasks (default priority 10)

Priority linux	Priority IEC
78	1
75	2
72	3
69	4
66	5
63	6

Low prior real time tasks

This tasks can be replaced at high workload and run in the linux scheduler "SCHED_FIFO".

Priority linux	Priority IEC
60	7

Priority linux	Priority IEC
57	8
54	9
51	10
48	11
46	Reserved for UOS task
45	12
42	13
39	14
36	15

Tasks outside real time

This tasks run in the linux scheduler "SCHED_OTHER".

Priority linux	Priority IEC
10 ("nice level")	16 - 31
0 ("nice level")	Default linux processes

7.2 Fast control

Fast control is used to reduce the system reaction time. This means reading the inputs, processing the data and writing the outputs need less cycles.

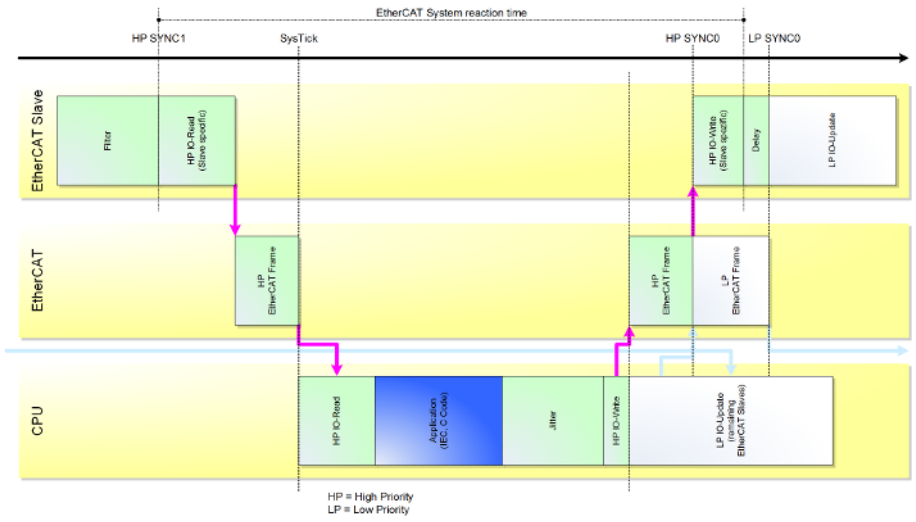


Fig. 7-5: Timing Diagram

The configuration of the fast control happens in the u-create studio (see online help "Configure EtherCAT slave").

7.3 Device Service

The DeviceService is a software service t manages, controls and monitors individual programs on the control. The DeviceService can collect and forward information (e.g. error messages) of the programs.

The DeviceService runs as an independent process and starts automatically once the operating system has been fully loaded.

Each program to be managed by the DeviceService must be defined on the control as `DeviceServiceItem`.

7.3.1 Defined state model

The DeviceService is responsible for software state transitions that do affect multiple software subsystems running on the same device.

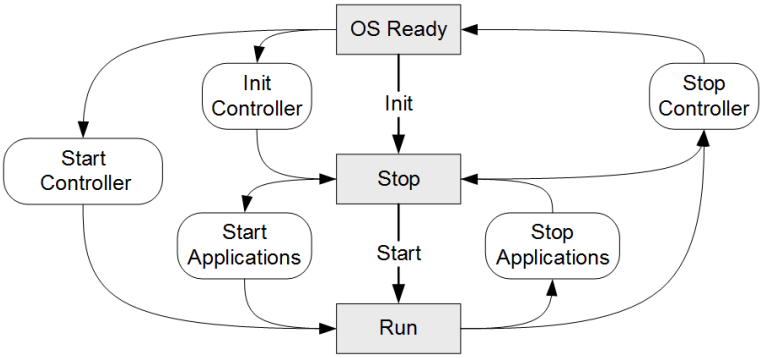


Fig. 7-6: DeviceService state model

States

State	Description
OsReady	The operating system is ready and the Device Service is available. Programs not initialized.
Stop	The programs are running, Applications not running
Run	Programs and applications are running

State transitions may be triggered explicitly by software components running on the local device, the operating elements on the device, or by remote clients (e.g. programming tools).

The following commands can be performed by the DeviceService for the state transition of multiple programs on the current device:

Command	Command type	Description
Start Controller	State transition	This status transition is run automatically once the device is started. The status changes from "OsReady" to "Run".

Command	Command type	Description
Stop Controller	State transition	The status changes from "OsReady".
Start Applications	State transition	The status changes from "Stop" to "Run".
Stop Applications	State transition	The status changes from "Run" to "Stop".
Init Controller	State transition	The status changes from "OsReady" to "Stop".
Shutdown Controller	Special state transition	Closes all running software applications on the device and stops it.
Reboot Controller	Special state transition	Closes all running software applications on the device and restarts it.
Delete Applications	Comprehensive Command	Deletes all applications on the device ("reset to factory settings").
Restart Controller	Combined state transition	Execute "Stop Controller" followed by "Start Controller".
Restart Applications	Combined state transition	Execute "Stop Applications" followed by "Start Applications".

7.3.2 Device Service Item

A Device Service Item represents a program installed on the current device. To do this, a description file called `<itemname>.item` is created in the directory `/opt/deviceservice/items.d/`. The file must be created in Lua (http://www.lua.org). The program must implement a defined status model with defined transitions.

7.3.2.1 Defined state model Device Service Item

Each program (`DeviceServiceItem`) must implement a status model defined as given below to be managed by the DeviceService.

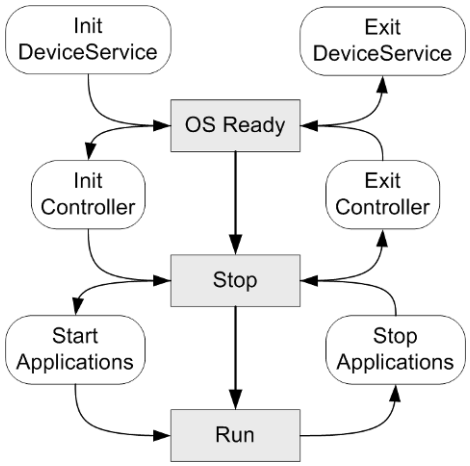


Fig. 7-7: DeviceService Item status model

State transitions of the Device Service state model are forwarded to all Device Service Items. So they can perform all actions required to execute their corresponding programs into the requested state.

States

The program must respond to the following status transitions:

Status transition	Description
Init DeviceService	The DeviceService is started and imports all programs created (DeviceService Items)
Init Controller	The program must execute all the necessary initializations and then change to the Stop process
Start Applications	The program begins processing and must change to the Run status
Stop Applications	The program ends processing and must change to the Stop status
Exit Controller	The program must end all initializations carried out in "Init Controller" and change to the OS-Ready status.
ExitDeviceService	The DeviceService ends all programs and itself

Commands that are forwarded to the programs can be found in chapter "Defined State Model".

7.3.2.2 Creating a description file

Each program to be managed by the DeviceService must be created containing the following elements and the functions described above, via a description file.

Name	Description
name	Name of the program (`DeviceServiceItem`)
variables	Global variables of the program
defaultvariables	Global variables with default values that can be overwritten

The system provides the following global variables (defaultvariables) with default values that can be overwritten by the program:

Global variable	Default value	Description
systemPath		Directory where the control software is stored.
applPath	/opt/kecontrolapplication/	Directory where applications are created and loaded
workPath	/opt/kecontrolapplication/	Directory in which the application can store data
autorestart	false	Defines whether or not the system restarts following an error

Global variable	Default value	Description
starepPathTmp	/tmp/	Directory where the files of the status report are stored
maxStarepCount	5	Number of stored status reports
autostart	true	Defines whether or not the programs are to be run immediately after the control is started
autostartTimeout	5	Time in which the autostart can be canceled [s]

Each `DeviceServiceItem` can export configuration variables or use configuration variables exported by other `DeviceServiceItem`. Variables are exported by adding them to the global `variables` variable:

```
variables = {
    someVar = 12,
    otherVar = 'text'
}
```

Example file `example.item.ex`:

```
item = {
     name = "example"

    variables ={
    }

    defaultvariables = {
    }
    ---------TESTTRANSITION-----------------
        testtransition = function(transition)
        --Init DeviceService--
        if transition == "Init DeviceService" then
            return true
        --Exit DeviceService--
        elseif transition == "Exit DeviceService" then
            return true
        --Init Controller--
        elseif transition == "Init Controller" then
            return true
        --Exit Controller--
        elseif transition == "Exit Controller" then
            return true
        --Start Applications--
        elseif transition == "Start Applications" then
             return true
        --Stop Applications--
        elseif transition == "Stop Applications" then
            return true
        end
        return false
    end

    , --- DOTRANSITION --------------------------
    dotransition = function(transition, step)
        --Init DeviceService
        if transition == "Init DeviceService" then
            if step == 1 then
            elseif step == 2 then
            elseif step == 3 then
            elseif step == 4 then
            elseif step == 5 then
            elseif step == 6 then
            elseif step == 7 then
            elseif step == 8 then
```

```
        elseif step == 9 then
        elseif step == 10 then
        end
--Exit DeviceService--
elseif transition == "Exit DeviceService" then
        if step == 1 then
        elseif step == 2 then
        elseif step == 3 then
        elseif step == 4 then
        elseif step == 5 then
        elseif step == 6 then
        elseif step == 7 then
        elseif step == 8 then
        elseif step == 9 then
        elseif step == 10 then
        end
--Init Controller--
elseif transition == "Init Controller" then
        if step == 1 then
        elseif step == 2 then
        elseif step == 3 then
        elseif step == 4 then
        elseif step == 5 then
        elseif step == 6 then
        elseif step == 7 then
        elseif step == 8 then
        elseif step == 9 then
        elseif step == 10 then
        end
--Exit Controller--
elseif transition == "Exit Controller" then
        if step == 1 then
        elseif step == 2 then
        elseif step == 3 then
        elseif step == 4 then
        elseif step == 5 then
        elseif step == 6 then
        elseif step == 7 then
        elseif step == 8 then
        elseif step == 9 then
        elseif step == 10 then
        end
--Start Applications--
elseif transition == "Start Applications" then
        if step == 1 then
        elseif step == 2 then
        elseif step == 3 then
        elseif step == 4 then
        elseif step == 5 then
        elseif step == 6 then
        elseif step == 7 then
        elseif step == 8 then
        elseif step == 9 then
        elseif step == 10 then
        end
--Stop Applications
elseif transition == "Stop Applications" then
        if step == 1 then
        elseif step == 2 then
        elseif step == 3 then
        elseif step == 4 then
        elseif step == 5 then
        elseif step == 6 then
        elseif step == 7 then
        elseif step == 8 then
        elseif step == 9 then
        elseif step == 10 then
        end
    end
, --- DOCHECK ------------------------------------
docheck = function()
end
```

```
    , --- DOSTATEREPORT ----------------------------
    dostatereport = function(stareppath)
    end

    , --- DOCLEARRETAIN ----------------------------
    doclearretain = function()
    end

    , --- DODELETEAPPLICATIONS ---------------------
    dodeleteapplications = function()
    end
}
```

7.3.3 Defined operation behaviour

Each `DeviceServiceItem` (program) can provide the following functions:

Function	Inputs	Return value	Description
testtransition	Status transition	true / false	Used to test whether the required status transition can be carried out by the program.
dotransition	Status transition, step number	-	Execute one step of a certain transition.
docheck	-		Checks that the program is working correctly.
dostatusreport	Path		Adds files to the status report currently triggered.
doclearretain	-		Deletes process of the retain files of the applications managed by the program.
dodeleteapplication	-		Deletes process of applications managed by the program.

Before each status transition, the DeviceService calls the `testtransition` function with the required status transition. This checks that the `DeviceServiceItem` is able to change its status. The function must return either `true` or `false`. No other value is permitted.

The DeviceService then calls the function `dotransition` with the status transition and a step number being carried out. Each `DeviceServiceItem` must execute the status transition. If an error occurs in this process, the `DeviceServiceItem` must call the Lua standard function `error` to be able to enter the error message with the DeviceService. The `dotransition` function is not permitted to have a return value.

The DeviceService checks for error messages. If an error has occurred in the `DeviceServiceItem`, the entire system is powered down. Depending on the configuration, the control automatically starts back up in the Run status.

The DeviceService performs a cyclical check by calling the `docheck` function to ensure that the registered `DeviceServiceItem` is working correctly.

The status report is a collection of information of all programs running on the control at a specific point in time. When a status report is triggered, all `DeviceServiceItem` are informed so that their information can be appended to the status report.

7.3.4 Status report

The status report is a collection of information software sub-systems running on a device at a specific point in time. If a status report is triggered, all `DeviceServiceItem` are informed so that this information can be added to the status report.

7.3.5 Co-routine model

`DeviceServiceItem` callbacks are run as Lua co-routines. This makes it possible to delay the running of a `DeviceServiceItem` by calling `coroutine.yield()`. This is preferable to a delay loop in the Lua script or intermediate solutions such as calling sleep functions using the operating system library.

Other co-routine functions are not permitted to be used.

8 Licensing

In some cases, Weidmüller products must be licensed in order enable use of the full scope of functions. By default, products are delivered with a "trial license". This means that a product's complete scope of functions can be used in full for a defined, limited period. During this period, a valid license must be requested from Weidmüller in order to be able to use the purchased functions for an unlimited period. To do so, please contact Weidmüller.

The following license types are possible:

- Single-user license for software products on the PC. This only applies to the PC where the software product was installed.
- Runtime license for software products on the device. These only apply to the devices where the software was installed.
- Trial license: License with limited validity period.

To license a Weidmüller product, a valid license ticket and a PC with an Internet connection are always required.

Information

Never get rid of your license ticket! The ticket is absolutely necessary in order to activate the license or deactivate/reactivate it in case of a device replacement.

In order to license the software product on the currently installed device (PC or control), the license (from the license ticket) must be activated. A license can only be activated once. Afterward, the software is bound to that device and can only be used in its full scope there. If device replacement is necessary in case of service, the activated license can be unlocked and then reactivated in the event of a PC replacement. When replacing a device, it is not possible to return the license, but the license can be restored on the new device. This is possible up to 2 x. Afterward, Weidmüller must be contacted.

8.1 Trial license

By default, all products are delivered with a trial license. This license expires after 30 days. Within this period, all product functions can be used in their full scope. After this time has expired, use is no longer possible, or only possible to a limited extent. For full use, a new, valid license is required, which can be requested from Weidmüller.

A trial license can only be used once. Even if the product is uninstalled and then reinstalled, use without a valid license is no longer possible.

8.1.1 Activation of a trial license (single user license)

When using a product on a PC, no activation of the trial license is necessary. Once the product has been installed, the trial license is automatically activated and the product can be used for 30 days from the time of installation.

8.1.2 Activation of a trial license (runtime license)

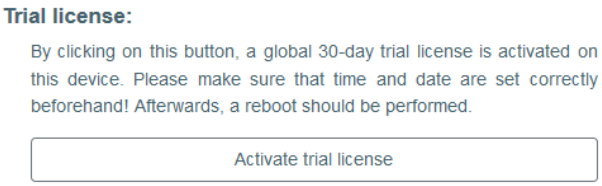
When using a device, the trial license must be activated manually. Otherwise, the product cannot be used or can only be used incompletely. Activation is only possible via the access "DevAdmin".

Information

Before activation, the date and time must be set correctly on the device! Otherwise, the license may become invalid immediately when the date is updated.

To activate the trial licence proceed as follows:

1) Start "DevAdmin"
2) Establish a connection to the desired device.
3) Switch to tab **Licensing ► Activation**.



4) Click on **Activate trial license**.

The trial license has been activated and the product is fully usable for 30 days.

8.2 Activating a single-user license (software on the PC)

The following options are available for activating a license on the PC:

- Online activation

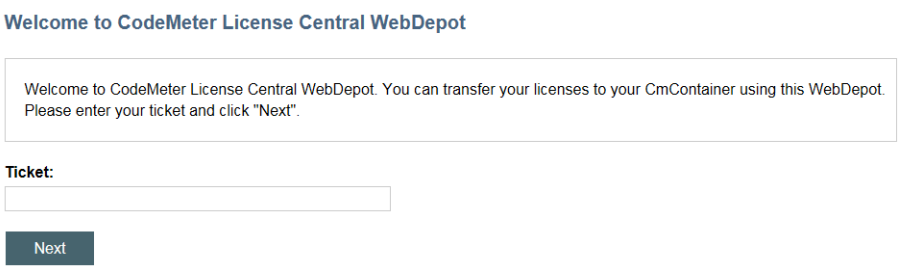
To activate the license, the "CodeMeter" license management tool from WIBU Systems (https://www.wibu.com/de.html) is required on the PC of the tool to be licensed. The tool is automatically installed during setup. (If the tool has been uninstalled manually, it can be re-downloaded on the manufacturer's side and reinstalled.)

Online activation

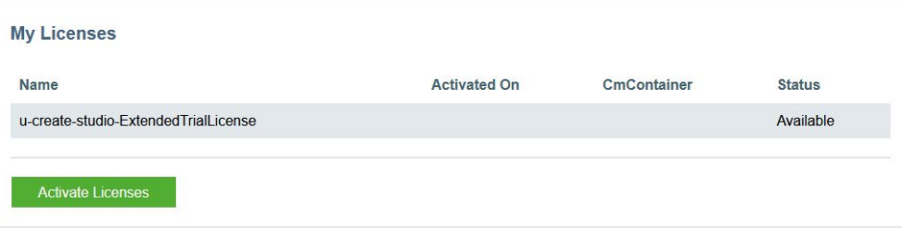
If the PC with the installed tool to be licensed has an Internet connection, online activation is possible. Activation takes place on this PC.

To do so, proceed as follows:

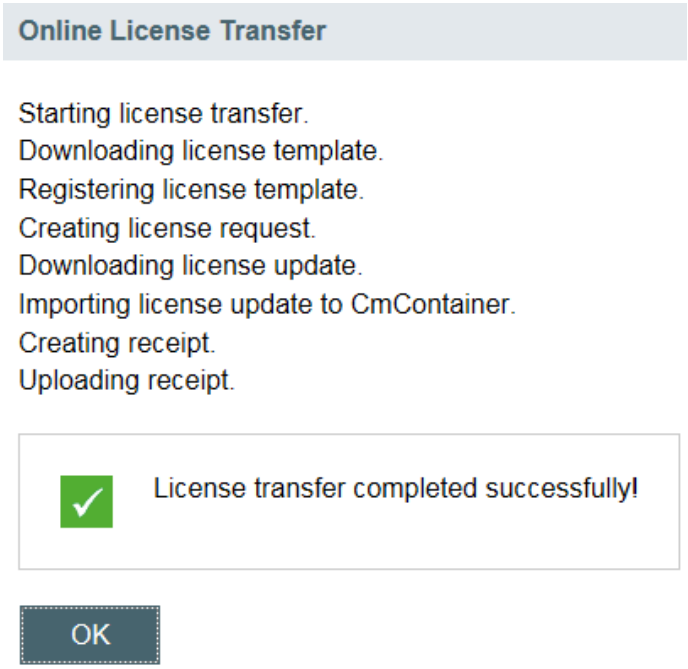
1) Open "WebDepot" by entering the link listed on the license ticket into an Internet browser on the PC.
2) Enter the ticket code and click on **Next**.



3) Click on **Activate Licenses**.



4) The following dialog opens.



5) Confirm the dialog with OK.

The license has been successfully activated.

8.3 Activating a runtime license (software on the device)

Folgende Möglichkeiten zur Aktivierung einer Lizenz am Gerät stehen zur Verfügung:

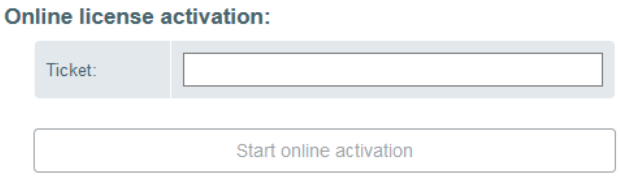
- Online-Aktivierung über "DevAdmin"

Online-Aktivierung über "DevAdmin"

Für diese Online-Aktivierung wird ein PC mit Internetverbindung benötigt. Das Gerät, auf dem die zu lizenzierende Software installiert ist, benötigt keine Internetverbindung, muss jedoch über eine Netzwerkverbindung mit dem PC verbunden sein.

Zum Aktivieren der Lizenz gehen Sie wie folgt vor:

1) Verbindung herstellen zwischen PC und Gerät mittels **DevAdmin**.
2) Klicken auf **Licensing ► Activation**.



3) Eingeben des Ticketcodes.
4) Klicken auf **Start online activation**.

Die Lizenz wurde am Gerät wurde erfolgreich aktiviert.

Information

Es wird empfohlen, ein Backup des Geräts zu sichern.

8.4 License status and license overview

Single-user licences

The status of a software license can be queried by clicking the icon in the task bar. Furthermore, a licensing overview can be called up using the link to the **WebAdmin**. In the **License monitor** tab, the license number and the number of assigned and available licenses are displayed. A license overview of all licenses and corresponding validity periods is located in the **Container** tab.

Runtime licenses

In order to query the status of a hardware license (runtime license), the device has to be accessed via "DevAdmin". In the **Licensing ► Overview** tab, an overview of all licenses and their validity periods as well as the license type is shown.

8.5 Device replacement

If device replacement is necessary in case of service, the activated license can be unlocked and then reactivated in the event of a PC replacement. When replacing a device, it is not possible to return the license, but the license can be restored on the new device. This is possible up to 2 x. Afterward, Weidmüller must be contacted.

8.5.1 PC replacement (single-user license)

The following options are available in which a license that has already been activated can be transferred to a new PC:

- Moving the license (Rehost): The existing PC is still functional. The license can be moved, i.e. returned to the old/existing PC and reactivated on the new PC.
- Reactivation the license (restore): The existing PC is no longer functional and can no longer return the license.

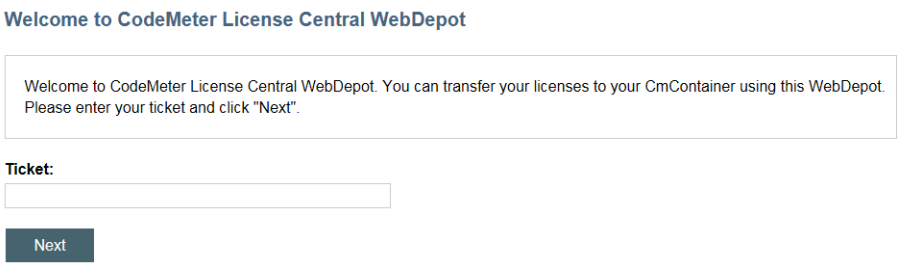
In any case, the PCs must have an Internet connection.

Moving the license (Rehost)

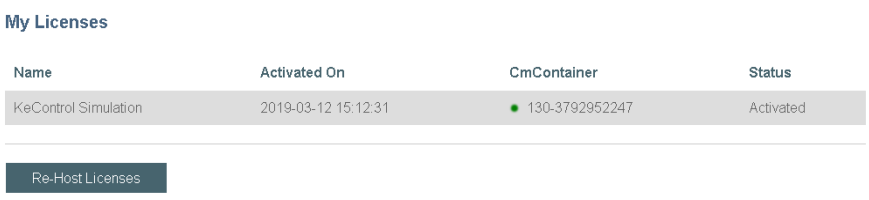
Using this option, a license is transferred from an old, functional PC to a new PC. Both PCs have Internet access.

In order to transfer a software license that has already been activated to a different PC, proceed as follows:

1) Open "WebDepot" by entering the link listed on the license ticket into an Internet browser on the old PC.
2) Enter the ticket code and click on **Next**.



3) Click on **Re-Host Licenses**



4) If necessary, select the license for deactivation.
5) Click on **Deactivate Selected Licenses Now**.
6) Confirm the dialog with OK.

The entered license is available for licensing again and can be used as described in Chap. "Licensing software on the PC".

Reactivation the license (restore)

In this case, the PC with the activated license is no longer functional, so returning the license is not possible. The new PC has Internet access.

To be able to reuse the license, first contact the Weidmüller Service department. The Weidmüller Service department authorizes the repeat activation of the license.

In order to activate the reactivated license on a new PC, proceed as follows:

1) Open "WebDepot" by entering the link listed on the license ticket into an Internet browser.
2) Enter the ticket code and click on **Next**.
3) Click on **Re-Store Licenses**.
4) Accept the restore conditions.
5) Click on **Restore Selected Licenses Now**.

The reactivated license was successfully activated on the PC.

Information

*Depending on the system, the button **Re-Host Licenses** is always displayed in the Web Depot. Clicking on this button triggers an error message, i.e. **Re-Host Licenses cannot** be used to **reactivate** the license.*

8.5.2 Device replacement (runtime license)

If a device must be replaced due to a defect, a backup of the old device must be restored on the new device. Only then the already activated license can be transferred to the new device. For this purpose the following possibility is available:

- Online restore of a license using DevAdmin

The license can only be used 2 x on a new device.

Information

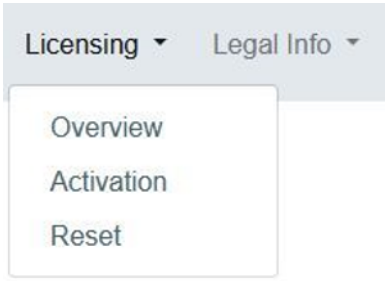
Only activated unlimited licenses can be transferred. If there is already an activated license on the new device, it will not be overwritten by the restore.

Online restore of a license using DevAdmin

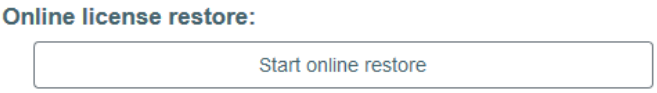
For restoring a license online, there is a PC with an Internet connection in the network of the device and it is running. The device to be licensed is also running.

1) Establish a connection between the PC and the device using **DevAdmin**.

2) Click on **Licensing ► Activation**.



3) Click on **License ► Online Restore**.
4) Click on **Start license Restore process**.



5) Confirm the dialog with **OK**.

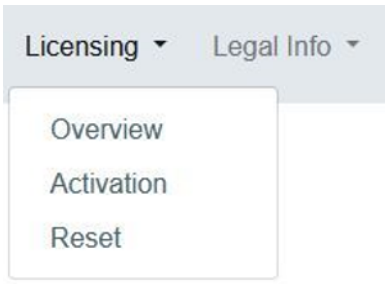
The entered license is available for licensing again and can be used as described in Chap. "Licensing software on a device".

8.6 Reset the license information

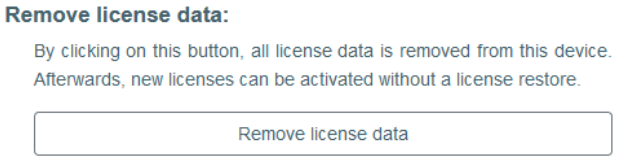
In certain cases it is neccessary to reset all license information of a device (e.g. if a restore fails). The license previously used on the device gets invalid and can no longer be reactivated.

To reset all license information of a device proceed as follows:

1) Establish a connection between the PC and the device using **DevAdmin**.
2) Click on **Licensing ► Reset**.



3) Click on **Remove license data**



4) Confirm the dialog with **OK**.

The license information of the device has been reset. To re-license the device, a new license must be purchased. Please contact the Weidmüller service.

9 Software Service

The Software Service tool is automatically installed with u-create. After successful installation, the icon appears in the right area of the Windows taskbar  **Software Service**.

The software service is used to store the components that can be loaded onto the controller.

More detailed information about the software service can be found in the online help.

System requirement

Port 1744 must be open. From version 2.5.0, this port is automatically released by the installation of the Software Service via the Windows firewall. The user needs the necessary Windows rights for this. If you do not have these rights, contact the administrator.

10 Device Administration (DevAdmin)

If a PC is within the network of the control, the login dialog for the **DevAdmin** can be opened on it via opening a internet browser and inserting the address `http://<IP address of the control>`.

The user is logged in with the following login data:

Username: Administrator

Password: tobechanged

The **DevAdmin** has the following tabs:

- **Diag.**: Read informationen of the control, trigger state report or crash report
- **Config.**: Perform network, time, and time zone settings
- **Backup/Restore**: Create or import a backup
- **Software Overview**: Overview of the devices in the system network
- **Licensing**: Perform licensing, see 8 Licensing
- **Legal Info**: Display of (open source) license texts of all installed software packages
- **Plugins**: Display of installed extensions

10.1 Tab "Diag."

This tab is divided into the following areas:

- Device Information
- Device State
- Statereport
- Crashreport

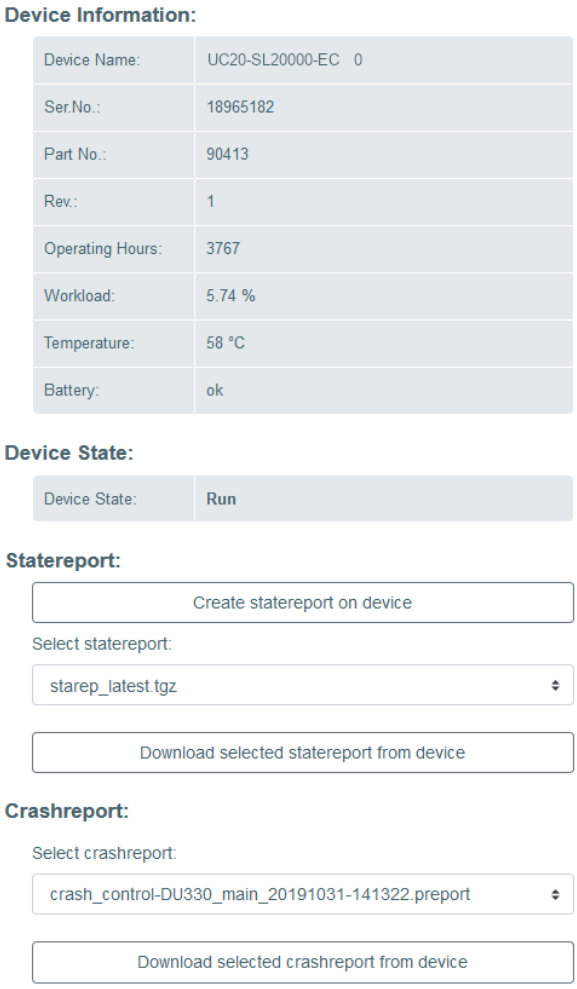


Fig. 10-8: DevAdmin - Diagnostics

Area "Device Information"

In this area the following information of the used control are displayed:

Information	Description
Device Name	Designation of the control
Ser.No.	Serial number of the control
Part No.	Material number of the control
Rev.	Revision number of the control
Operating Hours	Operating hours of the control
Workload	Current utilization of the control
Temperature	Current temperature of the control
Battery	State of the battery

Area "Device State"

In this area the following information of the used control are displayed:

Information	Description
Device State	Status of the control

Area "Statereport"

In this area a state report can be triggered via **Create statereport on device**. The report is saved under `/masterdisk/protocol/statusreport`. The report also contains the diagrams from the long-term recording ("Monitor" tab).

Via the dropdown menu a state report can be selected and downloaded to the PC.

Area "Crashreport"

Via the dropdown menu a crash report can be selected and downloaded to the PC.

10.2 Tab "Config."

This tab is divided into the following areas:

- Network Hostname
- Interface Ethernet X1
- Time, Date and Timezone
- Software Service
- Software Unit Key

Network Hostname

In the area "Network Hostname" the name oft the contol is entered.

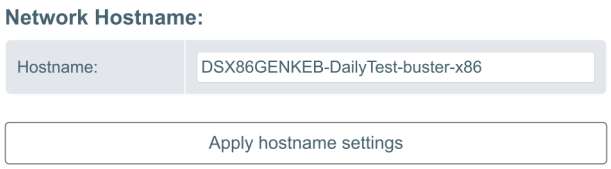


Fig. 10-9: Network Hostname

Setting	Description
Hostname:	Name of control
Apply hostname settings	The hostname of the control is applied.

Information

Changing the host name may cause connection problems of the control.

Interface Ethernet X1

In this area the IP address of the Ethernet interface X1 is configured.

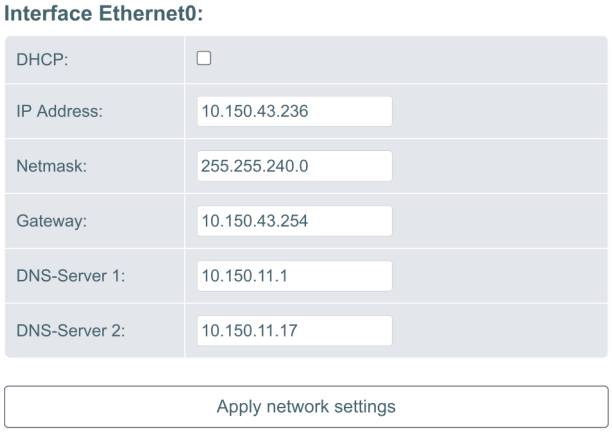


Fig. 10-10: Interface Ethernet X1

Seetting	Description
DHCP:	Activate or deactivate DHCP
IP Address:	IP address of the control
Netmask:	Set subnet mask
Gateway:	Set gateway
DNS-Server 1:	Set DNS server 1
DNS-Server 2:	Set DNS server 2
Apply network settings	The configuration of the interface Ethernet X1 are applied.

Time, Date and Timezone

In the area "Time, Date and Timezone" date, time and timezone are set.

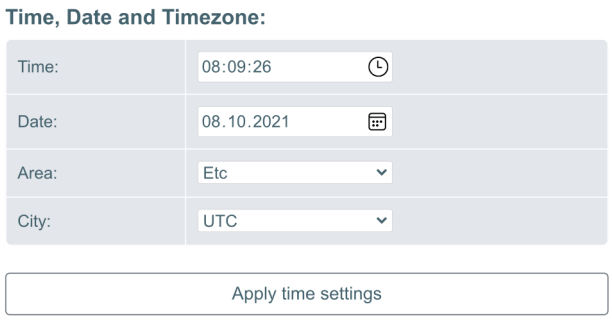


Fig. 10-11: Time, Date and Timezone

Setting	Description
Time:	Set clock
Date:	Set date

Setting	Description
Area:	Select region
City:	Select city
Apply time settings	Settings for time, date and time zone are applied.

Software Service

In the area "Software Service" the connection data to the software service is configured.

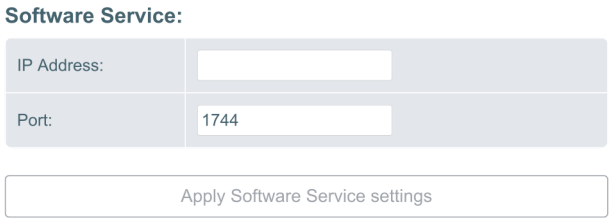


Fig. 10-12: Software Service

Setting	Description
IP Adress:	IP adress of PC the software service is installed
Port:	Network port of the software service (Standard: 1744)
Apply Software Service settings	Connection settings are applied and tested

Information

To accept and test the connection settings, the software must be started on the PC.

Software Unit Key

In the area "Software Unit Key", the private key required for decryption is transferred to the control.

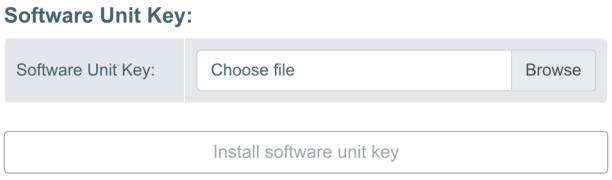


Fig. 10-13: Software Unit Key

Setting	Description
Software Unit Key:	Select private key file

Setting	Description
Install software unit key	Private key is installed on control

10.3 Tab "Backup/Restore"

The tab "Backup/Restore" is devided into the following areas:

- Backup
- Restore

Backup

In the area "Backup" backup copies can be created. Depending on the selection, different fields are active.

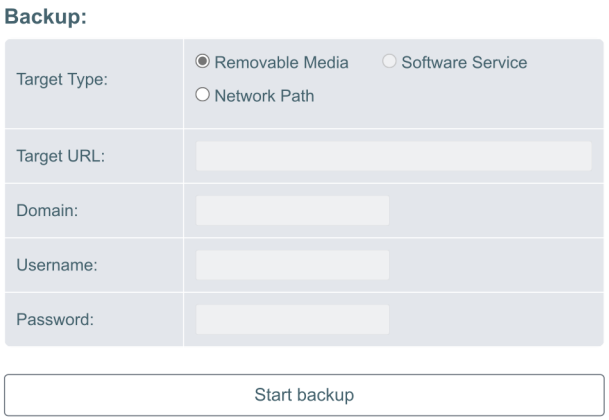


Fig. 10-14: Backup

Setting	Description
Target Type:	• Removable Media: Removeable media • Software Service (only available, if the software service has been configured) • Network Path: Path to a storage location on the network
Target URL:	Path to a storage location on the network
Domain:	Domain (optional)
Username:	User name
Password:	Password
Start backup	Backup copy is created. A confirmation prompt appears. Preparation begins and the device is restarted. No further information is displayed via the web interface.

Restore

In the area "Restore" copies can be restored. Depending on the selection, different fields are active.

Information

*If **Removable Media** is choosen, the backup copies are stored in a directory named **images**.*

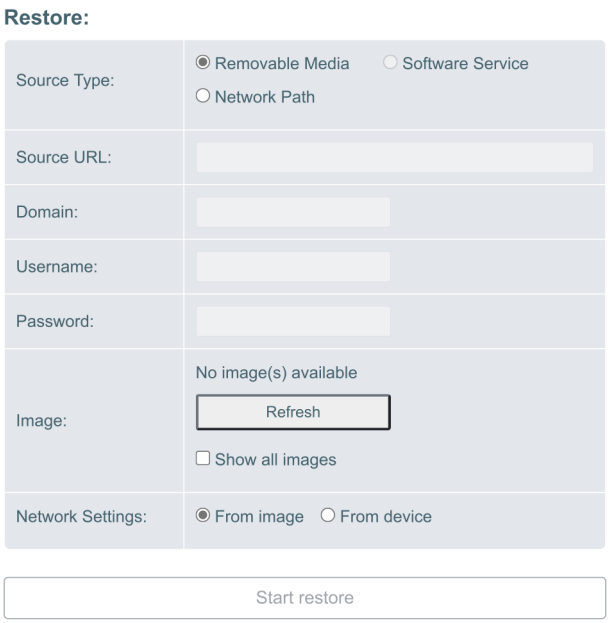


Fig. 10-15: Restore

Setting	Description
Source Type:	• Removable Media: Removeable media • Software Service (only available, if the software service has been configured) • Network Path: Path to a storage location on the network
Source URL:	Path to a restore location on the network
Domain:	Domain (optional)
Username:	User name
Password:	Password
Image:	The found backup copies are listed automatically. Refresh: selection box is updated Show all images: show all backup copies
Network settings:	• From image: Network settings are taken from the backup copy to be restored • From device: Current network settings of the device are kept
Start restore	Backup copy is restored. A confirmation prompt appears. Preparation begins and the device is restarted. No further information is displayed via the web interface.

10.4 Tab "Plugins"

In the "Plugins" tab, the extensions installed later are displayed.

This chapter describes how to create a plugin. Then the example of the monitor plugin is used to explain which information can be retrieved in the "Plugins" tab.

10.4.1 Add plugin to DevAdmin

There are the following ways to add own content under the Plugins tab:

- Content as HTML files
- Content hosted on a separate server

The files to be displayed under the "Plugins" tab must be stored on the control in the directory `/opt/devadmin/plugins/<name>`. There must always be a configuration file named `config.json`. In this file necessary configurations are specified.

Information

A data exchange between DevAdmin and own content is not possible.

Specification of configuration file

The file `config.json` contains the following configuration entries.

Entry	Type / Value	Description
`pluginType`	`iFrame`	Plugin type
`iFrame.type`	`file` or `link`	Type of integration (`file`: HTML files, `link`: server)
`iFrame.path`	STRING	For `file`: Relative path specification of `index.html` from base directory
`iFrame.url`	STRING	For `link`: URL of server
`iFrame.port`	0 - 65535	For `link`: Port of server
`fullScreen`	BOOL	Not available yet
`displayName`	STRING	Display name in the dropdown list under "Plugins"

Content as HTML files

The start file must have the name `index.html` and be specified in the configuration file. Structure of the configuration file `config.json`:

```
{
    "pluginType": "iFrame",
    "iFrame": {
        "type": "file",
        "path": "html/index.html"
    },
    "fullScreen": false,
```

```
    "displayName": "My new Plugin"
}
```

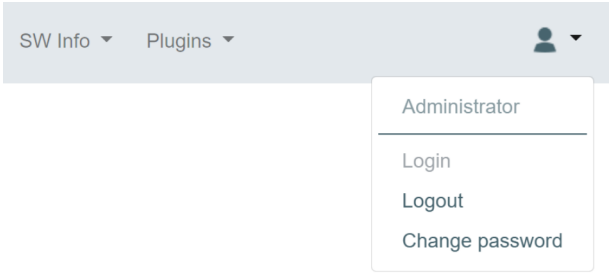
Content hosted on a separate server

Structure of the configuration file `config.json`:

```
{
    "pluginType": "iFrame",
    "iFrame": {
        "type": "link",
        "url": "https://myserver.com/",
        "port": 1880
    },
    "fullScreen": false,
    "displayName": "Server Plugin"
}
```

10.5 Change password for DevAdmin

At the top right of the **DevAdmin** is the user menu which contains the **Change password** entry.



If this is clicked, the window opens in which the password for the **DevAdmin** can be changed.

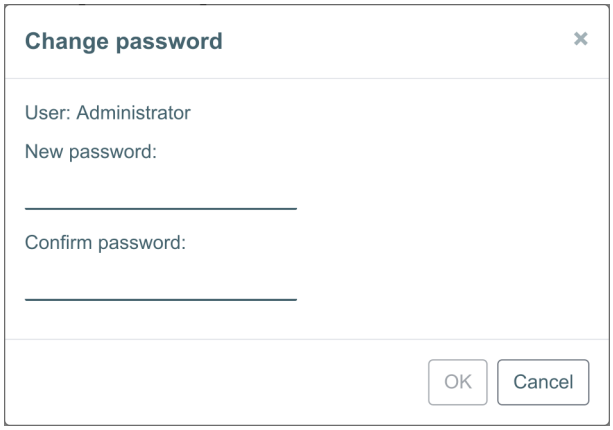


Fig. 10-16: Change password

Setting	Description
User:	Username

Setting	Description
New password:	Enter new password
Confirm password:	Confirm the new password
OK	Password is applied
Cancel	Process is aborted

11 OPC UA Server

The OPC UA Server can be installed as an optional software package (see "Software Units"). This chapter describes the use of the OPC UA Server on Weidmüller controls with the Linux operating system.

OPC Unified Architecture (OPC UA) is a standard for manufacturer-independent networking of devices in the automation sector. This provides standardized access options to the control. This standard allows any program (e.g. an OPC UA client) that supports this standard to communicate with the Weidmüller control and access data. For general information see: www.opcfoundation.org

Overview

The OPC UA Server runs in addition to the applications on the control and allows OPC UA clients to connect.

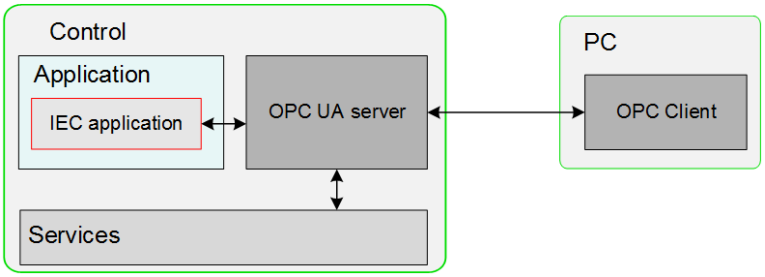


Fig. 11-17: Overview

Installation

The OPC UA Server can be installed on the control as an optional package via a target removable storage device. In the selection dialog, you can specify whether the OPC UA Server also accepts unencrypted connections.

For further information, please refer to u-create studio online help.

11.1 Overview of variants

The following table provides an overview of the product versions and their functionalities.

Functionality	OPC UA Server
SecureChannel:	x
Number of simultaneous client connections (sessions):	5
User authentication:	x
Services as per table below:	x
Generic variable tree:	x
Creating an information model:	-

Functionality	OPC UA Server
Support of customer-specific events and methods:	-

*) These functionalities are not currently supported yet.

The following table provides an overview of the service sets supported by the OPC UA server.

Service Sets as per OPC UA Specification Part 4	Supported services
Discovery	complete
SecureChannel	complete
Session	complete
Nodemanagement	not supported
View	• Browse • BrowseNext • TranslateBrowsePathsToNodeIds
Query	not supported
Attributes	• Read • Write
Method	not supported
MonitoredItem	• CreateMonitoredItems • ModifyMonitoredItems • SetMonitoringMode • DeleteMonitoredItems
Subscription	• CreateSubscription • ModifySubscription • SetPublishingMode • Publish • Republish • DeleteSubscriptions

11.2 Establishing a connection

The OPC UA Server supports the establishment of connections to OPC UA standard compliant clients via TCP-IP binary protocol. The IP address of the control is also the IP address of the OPC UA server. The port is 4842 by default. Application-specific settings such as port, application name, manufacturer, etc. can be set via the server configuration u-create studio.

Example of a connection call:

```
opc.tcp://ServerUrl:Port
```

Free and licensed OPC UA clients are available on the Internet for testing the functionality of the OPC UA Server. The endpoint URLs of the OPC UA Server returned by the Discovery service always contain the host name of the controller. The OPC UA Client must be able to resolve the host name when connecting to the Server IP (e.g. via host file or DNS).

Information

The host name must never begin with a digit, otherwise problems may occur during server boot-up or connection setup.

If the Discovery service returns an error code, there is either a network problem or it was not possible to start the server correctly. More detailed error sources can be found in Server Logging, see 11.6 Logging of server operation.

If the OPC UA Server finds expired licenses or no licenses at all during the connection setup (`ActivateSessionRequest`), the OPC UA error code `BadLicenseNotAvailable` is returned. It is not possible to set up a connection.

The OPC UA Server supports the following connection types:

- Unencrypted connection
- Encrypted connection via certificate

Unencrypted connection

If unencrypted connections have been configured at the time of installation, the OPC UA server can be accessed with the OPC UA client via an unencrypted connection. The registration can be done anonymously or by username and password. An anonymous client has full access to all nodes and network data.

In the user administration the user "Administrator" is created with the password "tobechanged" as standard. The OPC UA Server accepts all users created in the user administration.

Information

The use of an unencrypted connection during operation is strongly advised against as this represents a major security risk due to possible unauthorized access. The unencrypted connection should therefore only be used for development purposes.

Encrypted connection

With an encrypted connection, both OPC UA Server and OPC UA Client must authenticate each other. This is done via a "public-key" procedure using certificates. Basic256Sha256 (Sign, Sign & Encrypt) is used as the encryption algorithm. (For further literature on the certificate mechanism, see e.g. OPC Foundation Website.)

The OPC UA Server automatically creates the following directories on the controller in the directory `/opt/kecontrolapplication/OpcUa/PKI/CA`:

Directory	Description
`/own`	Contains the secret server key and the server certificate.
`/trusted/certs`	Contains all trusted client certificates.
`/trusted/crl`	Contains all revoked certificates (Certificate Revocation List). These clients can no longer connect.
`/issuers/certs`	Contains all trusted certificates used for verification.
`/issuers/crl`	Contains all revoked certificates (Certificate Revocation List), used for verification.
`/rejected`	Contains all client certificates rejected by the server. These clients cannot connect.

The OPC UA Server generates a server certificate (`uaserverc.der`) during startup. Instead of the automatically generated server certificate, you can also use your own generated certificate, which has to be stored in the directory `/opt/kecontrolapplication/OpcUa/PKI/CA/own`. This certificate must be known to each OPC UA client that wants to authenticate itself to the OPC UA server.

Only connection requests from OPC UA clients whose certificate is stored in the `/trusted/certs` directory are accepted. If a client connects to the server and its certificate is unknown, the connection setup is rejected and the rejected certificate is stored in the `/rejected` directory.

After the client and server have successfully authenticated themselves, user authentication with user name and password is also essential.

11.3 Server configuration

In this chapter the configurations of the OPC UA server are described. These can be made via "Expert entries" in the u-create studio. All OPC UA configurations must be listed in the `[OpcUa]` section. Configuration changes only become effective after restart.

Application-specific configurations

The following application-specific settings can be configured:

Designation	Description
Port	Server port
ApplicationName	Application name
ApplicationURI	Application URI
ProductURI	Product URI
ProductName	Product name
ManufacturerName	Manufacturer name

Designation	Description
ShowVariableTree	Display of generic variable tree (1 = activated, 0 = deactivated)
GenericVariablesNodeName	Name of generic variable tree (name is displayed in attributes "DisplayName", "BrowseName", "Description")
GenericEventsNodeName	Only available with unencrypted connection, internal parameter
GenericFunctionNodeName	Only available with unencrypted connection, internal parameter

Example

```
[OpcUa]
[OpcUa.Application]
Port=4840
ApplicationName="SampleApplicationName"
ApplicationURI="SampleApplicationURI"
ProductURI="http://www.company.com/"
ProductName="OPC-UA Server"
Manufacturer Name="Sample Manufacturer"
ShowVariableTree = 1 // 1 = activate, 0 = deactivate
```

"SecureChannel"configurations

With an encrypted connection, the Basic256Sha256 encryption algorithm is supported with 2 different setting options (`Sign` and `Sign and Encrypt`). All configuration values can be configured via integer values.

Security Policy	Value
Basic256Sha256	3

Message Mode	Value
Sign	1
Sign and Encrypt	2

Information

The "Basic256" and "Basic128Rsa15" encryption algorithms are no longer supported because they no longer meet current security requirements.

Example (default setting)

```
[OpcUa]
// Policy 3 = Basic256Sha256
// MessageMode 1 = Sign, 2 = SignAndEncrypt
[OpcUa.Security.CommunicationChannel.Communication:0]
Policy=3
MessageMode=1
[OpcUa.Security.CommunicationChannel.Communication:1]
Policy=3
MessageMode=2
```

Additional communication channels (`CommunicationChannel`) with different security settings can be added. Each additional connection is declared with an ascending number. The number must be unique. The `Policy` and `MessageMode` entries must always be set for each connection.

"Subscription" configuration

For the "Subscription" service, the supported time intervals can be configured. By default, 50 ms is set for the minimum and 3 600 000 ms for the maximum limit.

Possible configuration entries:

Name	Description
MinPublishingInterval	Minimum time interval in milliseconds, data type Double
MaxPublishingInterval	Maximum time interval in milliseconds, data type Double

Information

The minimum time interval chosen should not be too small, since the OPC UA server can reach a high load by answering these requests if there are several clients.

When sending a request (`CreateSubscription`), the client submits a desired time interval. The server checks whether the required interval is within the configured limits. If this is not the case, the limit value set is rejected by the server.

Example

```
[OpcUa.Subscription.Publishing]
MinPublishingInterval=100.50 // double
MaxPublishingInterval=10000.50 // double
```

The client requests an interval of 70 ms. The server responds that only a time interval of 100.5 is possible.

Recording values (samples)

The default recording intervals are set to 50 ms, 100 ms, 250 ms, 500 ms, 1000 ms, 2500 ms and 5000 ms. The intervals can be changed via the configuration, the data type is Integer:

```
[OpcUa.Subscription.Publishing]
[OpcUa.Subscription.Sampling.Intervals.Interval:0]
Interval=100 // int32 - ms
[OpcUa.Subscription.Sampling.Intervals.Interval:1]
Interval=20 // int32 - ms
[OpcUa.Subscription.Sampling.Intervals.Interval:2]
Interval=500 // int32 - ms
```

Several recording intervals can be defined by increasing the interval number. Every interval number must be unique.

When sending a request (`CreateMonitoredItem`), the client submits a desired recording interval. The server checks whether the required interval is within the configured limits. If this is not the case, the closest value set is returned by the server.

Example 1 (Standard configuration)

Standard recording intervals: 50 ms, 100 ms, 250 ms, 500 ms, 1000 ms, 2500 ms and 5000 ms

The client requests an interval of 70 ms. The server answers that the monitored object (`MonitoredItem`) recorded at intervals of 50 ms.

Example 2 (example configuration)

Configured recording intervals: 20 ms, 100 ms, 500 ms

The client requests a sampling interval of 70 ms. The server answers that the monitored item (`MonitoredItem`) recorded at intervals of 100 ms.

11.4 Generic variable tree

The generic variable tree is used to display the system variables of the IEC application and can be activated / deactivated via the expert entry `ShowVariableTree` (in u-create studio). By default, the generic variable tree is deactivated.

```
[OpcUa]
[OpcUa.Application]
ShowVariableTree = 1 // 1 = activate, 0 = deactivate
```

After activating the generic variable tree and restarting the controller, the generic variable tree is located in the directory `Root/Objects` (NodeId: Name space URI = "KeControlOpcUa/KeControlVar", Numeric ID = 0, DisplayName = "KeControl Variables").

The generic variable tree is an image of the KeStudioU4 u-create studio Variable Browser. The directory structure consists of the root nodes APPL, in which the approved IEC variables are located, and SYS, which contains the data of the system configuration. With the exception of SYS.CAT, all variable browser data are also visible in the OPC UA Server. The application and configuration data are displayed using object nodes (`FolderType`) and variable nodes. The node ID of the node is always of type String and is derived from the path of the data. All nodes are created in the "KeControlOpcUa/KeControlVar" namespace.

Example

The boolean application variable `VarBool` under APPL.system has the node ID "APPL.system.VarBool" and the OPC UA data type Boolean.

Arrays and structures are created in the generic variable tree using directories and individual variable nodes for the elements. All variable nodes offer read and write access (`Attribute AccessLevel`). Whether or not read and write operations work depends on the underlying data source. Approved system variables of the IEC in the APPL directory now support read and write access. Certain system configurations, however, cannot be overwritten. In case of an unauthorized write access, a corresponding error code is returned.

11.5 Creating an information model

The OPC UA Server supports the ability specified in the standard to create an information model via an XML file. This means that the OPC UA Standard address space can be expanded by adding a customer-specific information model. You can also define the structure for displaying the data.

All OPC UA information (variables, data types, methods, etc.) is displayed as nodes. Each node is assigned to a namespace. Namespaces and node IDs allow a unique identification. You can define your own namespaces, object and variable instances in the XML file. Methods, types (reference, object, variable, data types) or views are not supported.

The information must be defined in exactly one XML file and this file must be stored on the control in the directory `/appldisk/application/OpcUa/`. No other XML files may be located in this directory. When the server is started up, the XML file is read out and the defined information model is created. Adaptations in the XML file only become effective after restart.

If the XML file contains errors (e.g. incorrect syntax, incorrect data type assignment), the OPC UA server does not start. Clients cannot connect anymore. The cause of the error can be examined more closely in the System Trace. See chapter 11.6 Logging of server operation.

Namespaces

A namespace is uniquely identified by its URI. This URI corresponds to a numeric index assigned by the OPC UA server during startup. The namespace `"http://opcfoundation.org/UA/" (index = 0)` is already specified by the OPC UA standard and contains predefined data types and nodes for OPC UA server diagnostics.

Namespaces are described within the XML tag `<NamespaceUris>`. Several namespaces can be created there using `<Uri>`. The following attributes are permissible:

Attribute	Description
`ns`	This attribute is used to assign the numeric index of the namespace URI used within the XML file. Only namespace 0, predefined by the OPC UA standard, can be used without explicit XML configuration. The numeric index is reassigned by the OPC UA server at runtime. This index can be read out using the "NamespaceArray" object.
`isDefaultNs`	You can use this attribute to specify a namespace as the default value within the configuration file. When describing instances, this namespace no longer has to be specified explicitly in the definition of the NodeId. Only one standard namespace can be used.

In the following example, two namespaces are created:

```
<NamespaceUris>
    <Uri ns="1" isDefaultNs="1">www.company.com</Uri>
    <Uri ns="2">MyNamespace</Uri>
</NamespaceUris>
```

Object and variable instances

To display variables or methods in the OPC UA client, instances must be created and linked to the desired variables and methods on the control.

Information

Currently not all data types defined in the OPC UA standard are supported.

The following table describes the supported OPC UA data types with the corresponding IEC data types. The unique node address that must be used when creating in the XML file is specified for each data type. The node address consists of multiple so-called "BrowseNames". BrowseName is an OPC UA concept that can be used to specify node addresses. A BrowseName consists of a namespace index and a name. In the XML definition, these are separated by a colon. The BrowseNames are compiled into a node address in XML separated by a period. The node addresses for the standard data types are specified by the OPC UA standard and can be read out from the address space using a client.

The node address for a Boolean variable is defined as follows, for example: `0:Types.0:DataTypes.0:BaseDataType.0:Boolean`. In the table below, the node addresses are displayed in abbreviated form. However, for each node address, in the XML file the following character string must be appended at the front: `0:Types.0:DataTypes.0:BaseDataType.`

OPC UA data type	IEC data type	Node address
Boolean	BOOL	`0:Boolean`
SByte	SINT	`0:Number.0:Integer.0: SByte`
Byte	BYTE, USINT	`0:Number.0:UInteger.0:Byte`
Int16	INT	`0:Number.0:Integer.0:Int16`
UInt16	WORD, UINT	`0:Number.0:UInteger.0:UInt16`
Int32	DINT	`0:Number.0:Integer.0:Int32`
UInt32	DWORD, UDINT	`0:Number.0:UInteger.0:UInt32`
Int64	LINT	`0:Number.0:Integer.0:Int64`
UInt64	LWORD, ULINT	`0:Number.0:UInteger.0:UInt64`
Float	REAL	`0:Number.0:Float`
Double	LREAL	`0:Number.0: Double`
String	STRING, WSTRING	`0:String`
DateTime	DATE_AND_TIME, DATE, TIME_OF_DAY	`0:DateTime`

Information

The mapped data types between OPC UA and IEC must match, otherwise a node is not generated.

Information

The instantiation of structure variables or more complex (basic) data types is not supported.

The OPC UA data types `BaseObjectType (0:Types.0:ObjectTypes.0:BaseObjectType)` and `FolderType (0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType)` can be used to build a hierarchical model.

All instances are collected under a `Instances` entry. This entry needs an attribute, `Parent`, which defines the root node of the following instances as a node address. The `Instances` entry can be defined multiple times in succession. The root node must exist, as otherwise no child nodes can be generated.

The following example shows the instantiation of two directories within the `Objects` node predefined by the OPC UA standard. The node hierarchy is determined by the hierarchy of the XML entries.

```
<Instances Parent="0:Objects">
    <Object
        DataType="0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType"
        NodeId="1:i=1000" BrowseName="Folder" DisplayName="Folder">

        <Object
            DataType="0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType"
            NodeId="1:i=1001" BrowseName="Subfolder" DisplayName="Subfolder">

            <!-- define further objects or variables .... -->

        </Object>
    </Object>
</Instances>
```

For the object instantiation, the XML entry `Object` is used, while for variables, the XML entry `Variable` is used. The following attributes have to be specified for both tags:

- DataType
- NodeId
- BrowseName
- DisplayName

The description of the node IDs (NodeId) always consists of a namespace index together with the specification of an identifier. The identifiers have to be specified numerically (i=xx). Variables also need a value and, optionally, an access right (attribute `AccessLevel`). The value of the variable can either be set by a constant, which is defined by the `ValueConstant` attribute, or by the path of a system variable, defined by the `ValuePath` attribute.

Possible XML attributes:

Attribute	Description
`DataType`	Node address of the required data type for the instance. The node address of the types can be read out of the address space with the help of an OPC UA client, if necessary.
`NodeId`	Defines the node ID of the current instance, and this must be unique. This server version only supports numeric node IDs.
`BrowseName`	Internally used name for node addresses.
`DisplayName`	Displayed name for the OPC UA client.
`AccessLevel` (only variables)	Defines the access rights for the `Value` attribute of a variable. By default, no access is allowed. The attribute is specified as hexadecimal number and interpreted as `AccessLevelType` defined in the OPC UA specification 3. Example: "0x01" = CurrentRead, "0x03" = CurrentRead and CurrentWrite.
`ValueConstant` (only variables)	Allocation of a constant.
`ValuePath` (only variables)	Path to a system variable. The path can be displayed in the u-create studio browser after successful login.

In the following example, an OPC UA `Boolean`-type variable is instanced in the server address space and linked to a variable on the controller. When defining the Node ID, the namespace index was not specified in the example. This uses the default namespace (if `NamspaceUris` is defined in the entry).

```
<Variable DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
    NodeId="i=1002" BrowseName="MyBool" DisplayName="MyBool"
    ValuePath="APPL.system.vBOOL" AccessLevel="0x01"/>
```

Instead of a variable path, constants defined in the XML file can also be linked. All data types specified above are supported, except `DateTime`.

In the following example, an OPC UA `Boolean`-type variable is instanced and the value is linked to a constant:

```
<Variable DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
    NodeId="i=1003" BrowseName="MyBool" DisplayName="MyBool"
    ValuePath="true" AccessLevel="0x03"/>
```

Arrays

Information

The OPC UA Server supports only one-dimensional arrays with basic data types and variable mapping (`Attribute ValuePath`).

For instantiating arrays, another entry, `ArrayDimension`, and the following additional attribute is needed:

Attribute	Description
`ValueRank="1"`	Determines the number of array dimensions. Only the value "1" is supported.

In the following example, the entire IEC array is instantiated in the server as an OPC UA array.

```
<Variable DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
    NodeId="i=10" BrowseName="bool array" DisplayName="bool array"
    AccessLevel="0x03" ValueRank="1">
    <ArrayDimension ValuePath="APPL.system.vBoolArray"/>
<Variable/>
```

Optionally, a subrange of an array can be specified for variable mapping. The following attributes are required for this:

Attribute	Description
`ArrayOffset`	Specifies the start value of the array subrange (left index).
`ArrayLength`	Desired length of the array subrange.

In the following example, a subrange of the IEC array is instantiated in the server as an OPC UA array. In the IEC an array from 1 to 100 is defined. In the OPC UA server only the sector from 10 to 15 is mapped.

```
<Variable DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
    NodeId="i=10" BrowseName="bool array" DisplayName="bool array"
    AccessLevel="0x03" ValueRank="1">
    <ArrayDimension ArrayOffset="9" ArrayLength="5"
        ValuePath="APPL.system.vBoolArray"/>
<Variable/>
```

Example of a complete XML file

```
<OpcUAInformationModel>
<!-- Declaration of namespace URIs -->
<NamespaceUris>
    <Uri ns="1" isDefaultNs="1">http://www.company.com</Uri>
    <Uri ns="2">MyNamespace</Uri>
</NamespaceUris>

<!-- Declaration of instances -->
<!-- add child instances to parent Root/Objects node -->
<Instances Parent="0:Objects">

<!-- create folder My Variables -->
<Object DataType="0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType"
NodeId="1:i=1000" BrowseName="My Variables" DisplayName="My Variables">

<!-- add a UInt8 variable to My Variables folder -->
<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:UInteger.0:Byte"
NodeId="1:i=1003" BrowseName="MyUInt8" DisplayName="MyUInt8"
ValuePath="APPL.system.vUSINT" AccessLevel="0x03"/>
<!-- variable is read- and writeable -->

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:UInteger.0:UInt16"
NodeId="1:i=1004" BrowseName="MyUInt16" DisplayName="MyUInt16"
ValuePath="APPL.system.vUINT" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:UInteger.0:UInt32"
NodeId="1:i=1005" BrowseName="MyUInt32" DisplayName="MyUINt32"
ValuePath="APPL.system.vUDINT" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:UInteger.0:UInt64"
NodeId="1:i=1006" BrowseName="MyUInt64" DisplayName="MyUInt64"
ValuePath="APPL.system.vULINT" AccessLevel="0x03"/>
```

```
<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:SByte"
NodeId="1:i=1008" BrowseName="MyInt8" DisplayName="MyInt8"
ValuePath="APPL.system.vSINT" AccessLevel="0x01"/>
<!-- variable is only readable -->

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int16"
NodeId="1:i=1009" BrowseName="MyInt16" DisplayName="MyInt16"
ValuePath="APPL.system.vINT" AccessLevel="0x01"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int32"
NodeId="1:i=1010" BrowseName="MyInt32" DisplayName="MyInt32"
ValuePath="APPL.system.vDINT" AccessLevel="0x01"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int64"
NodeId="1:i=1011" BrowseName="MyInt64" DisplayName="MyInt64"
ValuePath="APPL.system.vLINT" AccessLevel="0x01"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Float"
NodeId="1:i=1013" BrowseName="MyFloat" DisplayName="MyFloat"
ValuePath="APPL.system.vREAL" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Double"
NodeId="1:i=1014" BrowseName="MyDouble" DisplayName="MyDouble"
ValuePath="APPL.system.vLREAL" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
NodeId="1:i=1016" BrowseName="MyBool" DisplayName="MyBool"
ValuePath="APPL.system.vBOOL" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:DateTime"
NodeId="1:i=1017" BrowseName="MyDateTime" DisplayName="MyDateTime"
ValuePath="APPL.system.vDATE_AND_TIME" AccessLevel="0x03"/>

<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:String"
NodeId="1:i=1018" BrowseName="MyString" DisplayName="MyString"
ValuePath="APPL.system.vSTRING" AccessLevel="0x03"/>

<!-- string constant -->
<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:String"
NodeId="1:i=1020" BrowseName="String Constant" DisplayName="String Constant"
ValueConstant="My test string constant" AccessLevel="0x03"/>

<!-- int32 constant-->
<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int32"
NodeId="1:i=1021" BrowseName="Int32 Constant" DisplayName="Int32 Constant"
ValueConstant="12" AccessLevel="0x03"/>

<!-- float constant -->
<Variable
DataType="0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Float"
NodeId="1:i=1022" BrowseName="Float Constant" DisplayName="Float Constant"
ValueConstant="12.47" AccessLevel="0x03"/>

<!-- create a subfolder for arrays -->
<Object DataType="0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType"
NodeId="1:i=1023" BrowseName="1-dim Arrays" DisplayName="1-dim Arrays">

<!-- boolean array - block mapping -->
<Variable DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean"
NodeId="i=1024" BrowseName="bool array" DisplayName="bool array"
AccessLevel="0x03" ValueRank="1"> <!-- ValueRank must always be 1 -->
<!-- ArrayOffset and ArrayLength are optional -->
```

```
<ArrayDimension ArrayOffset="0" ArrayLength="2"
ValuePath="APPL.system.vBoolArray"/>
</Variable>
</Object>
</Object>
</Instances>
</OpcUAInformationModel>
```

Role-based authorization model

The OPC UA Server supports the assignment of access rights for users based on user roles. These access rights only apply to the nodes defined in the XML file and when the client logs on with a user name and password. An anonymous client has full access to all nodes.

In the user administration, users and roles must be created and one or more roles assigned to the users. The role names in XML must match the names in the user administration. When an OPC UA client connects to the server, the server accesses the user administration to read the authentication data and roles of the client. Authorizations are assigned to the created roles in XML. These rights only apply to the nodes instantiated by the XML. If rights for roles have already been assigned in the user administration, these are not considered by the OPC UA Server. Before each request, the OPC UA Server checks whether the roles of the user meet the requirements for access rights.

If access to values of variable nodes is desired, the access permissions set for the respective variable node (`AccessLevel` attribute) are also checked beforehand. If, for example, the value of a variable node is not writable, the client cannot execute a write command, even if the authorizations are given based on the role assignment.

Global rights

Before variables and objects can be instantiated (XML `Instances` entry), roles and authorizations must be assigned globally. The permissions are defined as bitmasks and correspond to access type (`PermissionType`), defined in the OPC UA specification, Part 3.

With a terminating zero, a role name can only be a maximum of 128 characters long. A maximum of 30 different roles are supported. The authorizations are interpreted as hexadecimal numbers. The prefix "0x" is optional. Upper and lower case of the hex number (0xAABB or 0xaabb) is not taken into account.

Example

The following code extract shows the definition of 2 roles with rights. The role "Observer" can run through the tree and only has read access to the instanced nodes. The "Operator" role can also write.

```
<DefaultRolePermissions>
    <RolePermission RoleName="Observer" Permissions="0x21"/> <!-- Bit 0 -
Browse, Bit 5 - Read -->
    <RolePermission RoleName="Operator" Permissions="0x61"/> <!-- Bit 0 -
Browse, Bit 5 - Read, Bit 6 - Write -->
</DefaultRolePermissions>
```

A user without roles or with roles that were not defined globally in XML only has access to the standard address space of OPC UA and the variable tree. The same applies if no global role has been defined in XML. If a user has several roles, all the rights of the roles are taken into account.

Node-based XML rights

The globally declared authorizations of a role can be extended or restricted for each node.

The following example overwrites the authorization of the "Operator" role for the "Var1" variable node. In the global authorization definition "Browse", the role "Operator". can perform read and write accesses. For the "Var1" variable node the access to browse and read is now restricted.

```
<Variable
    DataType="0:Types.0:DataTypes.0:BaseDataType.0:Boolean" NodeId="i=5000"
BrowseName="Var1" DisplayName="Var1" AccessLevel="0x03" ValueRank="1">
    <NodeRolePermissions>
        <RolePermission RoleName="Operator" Permissions="0x21"/>
    </NodeRolePermissions>
</Variable>
```

11.6 Logging of server operation

The OPC UA server protocols information and errors about startup and execution at runtime. This information is either integrated into the general logging of the control and thus displayed in the trace monitor (in the u-create studio) or managed by the server as a separate file. This file is stored on the controller in the directory `/opt/kecontrol/protocol/OpcUa/`. The messages outputs here can provide useful information, in particular, for troubleshooting during information model creation (incorrect variable mapping, typing errors, already existing NodeIds, etc.).

Logging can be configured via expert entries (in u-create studio). If there is a general logging of the control, only the `TraceMask` parameter is taken into account. If a separate file is managed, the `TraceFileSizeKbMax` parameter can be additionally specified.

The value of the `TraceMask` configures the granularity of the output and is executed as a bit field. By default, the configuration is set to `TraceMask = 5`. The value to be set results from the bitwise addition of the following partial masks:

Bit	Partial mask	Logging
0	1	Startup information and error, configured by default
1	2	More detailed information and error output during startup
2	4	Runtime error (internal failures e.g. memory bottleneck, unexpected types for read/write etc.), configured by default
3	8	Detailed output about runtime errors

The `TraceMask` is interpreted as a decimal number. To activate all outputs, the `TraceMask` can be set to the value -1.

The `TraceFileSizeKbMax` parameter specifies the maximum file size in kB. The default value is 4096 kB. Once the maximum size has been reached, a further log file with suffix `.1` is created. The log files are then overwritten.

Example

```
[OpcUa]
[OpcUa.Trace]
TraceMask = 15
TraceFileSizeKbMax = 4096
```

12 Node-RED

Node-RED can be installed as an optional software package (see Software Units). It is a graphical development tool developed by IBM. This allows function modules, so-called "nodes", to be lined up in a certain sequence by drawing connections. Each function module has a defined task. Data can be processed and transferred between the function modules.

The graphical editor of Node-RED can be started via "DevAdmin" in the tab "Plugins". Further information on the functionality of Node-RED can be found in the internet. See also https://nodered.org/.

13 LongtermDiagnosticMonitor

The LongtermDiagnosticMonitor can be installed as an optional software package (see Software Units) and offers the possibility of long-term recording of data on the control. Via "Monitor" in the DevAdmin (tab Plugins) the data can be displayed.

13.1 Monitor

The monitor plugin is divided into the following areas:

- Problems
- Groups
- Categories

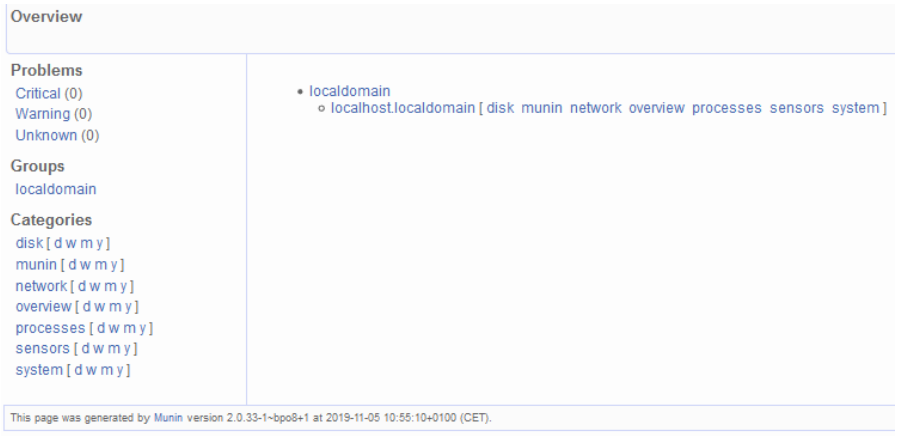


Fig. 13-18: DevAdmin - Monitor

Problems

In this area the occurring faults divided to their priotity are displayed.

Groups

In this area all available diagrams are listed and can be selected.

Categories

In this area the categories for the long-term diagnostic are listed. After choosing a category, the corresponding diagramms are displayed. These diagramms are also saved when triggering a status report.

Beside the category, the daily overview **d**, the weekly overview **w**, the montly overview **m** or the yearly overview **y** can be called via the corresponding letters **d w m y**. An overview with 4 diagramms consisting of daily, weekly, montly and yearly overview is opened through clicking on a diagram.

The following diagrams can be displayed:

Category	Diagram	Description
disk	Average latency for /dev/sda	Average waiting time (latency) for access to the directory `/dev/sda`. The waiting time shows how utilized the system ist. 1 second waiting time corresponds to an utilization of 100 %.
	Disk IOs per device	Inputs and outputs on the data carrier per device
	Disk latency per device	Waiting time on the data carrier
	Disk throughput for /dev/sda	Throughput of the data carrier for the directoy `/dev/sda` [in Byte per second]
	Disk utilization for /dev/sda	Utilization of the data carrier for the directory `/dev/sda`. If an input/output takes 1 second, the data carrier is 100 % utilized.
	IOs for /dev/sda	Number of inputs/outputs (I/O requirements) per second and their average size [in kB] (in the diagramm corresponds 1 kB = 1000 Byte).
	Throughput per device	Throughput of inputs/outputs for directory `/dev/sda`
	Utilization per device	Utilization of directory `/dev/sda` [in percent]
munin	Munin processing time	Processing time of the four different Munin-processes (munin update, munin graph, munin html, munin limits) [in seconds]
	Munin update	Time needed by Munin-processes for collecting the recorded data [in seconds]
network	Firewall Throughput	Throughput of the firewall [received and forwarded packages per second]
	Netstat	TCP activity of all network interfaces. Number of active, passive, failed, re-booted and current performed TCP connections per second.
	ETH0 errors	Number of errors, package losses and collission on the ETH0 network interface.
	ETH0 traffic	Data traffic of the ETH0 network interface [in bits per second]
	ETH1 errors	Number of errors, package losses and collission on the ETH1 network interface.
	ETH1 traffic	Data traffic of the ETH1 network interface [in bits per second]

Category	Diagram	Description
overview	CPU utilization	Overall utilization of the CPU per core [in percent]
	Disk lifetime	Remaining lifetime of the flash storage media [in percent]
	Disk space	Used and free memory on the flash storage media [in MB]
	PLC baselib heap memory statistics	Used and free memory on the "BaseLib Heap" storage media [in MB]
	RAM usage	RAM utilization devided in occupied, free and available [in MB]
	System task performance on core n	Distribution of computing power (per core) for the different process groups [in percent]
	Temperature information	Temperature of CPU and mainboard [in degree Celsius]
	Uptime of PX-Package	Power-on time of control packages (PX package) [in days]
processes	Fork rate	Number of rebootet processes [per second]
	Number of threads	Number of threads
	Processes	Number of processes divided into their state
	Processes priority	Number of processes divided into their importance
	VMstat	Utilization of CPU time through processes [in milli = thousandth]
system	Available entropy	Number of available random numbers
	CPU usage	CPU utilization divided in different process groups
	File table usage	Number of opened files and maximum number of opened files
	Individual interrupts	Number of interrupts divided in type of interrupt
	Inode table usage	Number of opened inodes
	Interrupts and context switches	Number of interrupts and context switches
	Load average	Average load, which shows the number of directly executable processes
	Logged in users	Logged-in users
	Memory usage	Display, which shows where the main memory is used [in bytes]
	Swap in/out	Transfer between main memory and swap-memory
	Uptime	Power-on time

14 Modbus Server

The Modbus server can be installed as an optional software package (see Software Units) and offers the possibility to read and write certain variables of the controller via Modbus.

The variables can be defined via the Expert configuration in u-create studio. Furthermore, the access rights and the conversion factor of a variable can be defined there.

The following interfaces are supported:

- Ethernet, Port 502

The following protocols are supported:

- Modbus TCP

Base library

The Modbus Server implementation is based on **libmodbus-3.1.4** under LGPL v2.1 license.

libmodbus – A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Win32, http://libmodbus.org/

Test Software Recommendation

QModMaster, https://sourceforge.net/projects/qmodmaster/

Python modbus-tk

14.1 Function description

The Modbus server is an executable program that is started with the controller. The Modbus server of the controller acts as server and processes the requests from a client. The external client has the possibility to read or change values.

Functions

The following functions are supported from the modus server:

Function code	Name
0x01	Read Coils
0x02	Read Discrete Inputs
0x03	Read Holding Registers
0x04	Read Input Registers
0x05	Write Single Coil
0x06	Write Single Register
0x0F	Write Multiple Coils
0x10	Write Multiple Registers

Function code	Name
0x16	Mask Write Register
0x17	Read / Write Multiple Registers

All other function calls are answered with an error message.

Error codes

Error name	Error number	Description
MODBUS_EXCEPTION_ILLEGAL_FUNCTION	1	Modbus function code not supported
MODBUS_EXCEPTION_ILLEGAL_DATA_ADDRESS	2	Variable for register / coil access is not configured
MODBUS_EXCEPTION_ILLEGAL_DATA_VALUE	3	• Number for access to several registers / coils is outside the specification • Number of bytes in the received Modbus command is invalid
MODBUS_EXCEPTION_SLAVE_OR_SERVER_FAILURE	4	• Modbus server configuration invalid • Control not running • No variable assigned for register / coil to be manipulated • Manipulation of several registers / coils beyond the configured address range • No rights for write access • No valid access to interleaved registers e.g. only 1 register is read with variable mapping to 2 registers • Mapping of an unsupported variable type

Data presentation

Modbus only supports 16 bit registers. A register can therefore only represent integer values without decimal places. Therefore, values represented as decimal numbers, unsigned integer values greater than 2^{16} (65535) or integer values with sign greater / less than 2^{15} (+/-32768) must be scaled by a conversion factor or distributed to several Modbus registers.

Coils work bit by bit. If the variable value is 0, the coil is not set. If the value is not 0, the coil is set. When writing coils, only 0 or 1 is written to the application variable.

14.2 Configuration

The configuration is done via "Expert Entries" in the project.

The configuration for the Modbus Server consists of the following sections:

- Basic configuration of the interface
- Configuration of the Holding Registers (optional)
- Configuration of the input registers (optional)
- Configuration of the coils (optional)
- Configuration of the Input-Coils (optional)

Each section consists of a base node in square brackets and sub-entries. The sub-entries can either be individual parameters or again a sub-node in square brackets. At the beginning of a sub-node is the parent node followed by a dot as separator. Each entry must be on a new line, but blank lines can be inserted. Indentations at the beginning are ignored. At the end of a line a comment text can be inserted starting with `//`.

14.2.1 Configuration interface

The name of the base node is `[ModBus]`.

The following parameters can be configured for the interface.

Selection of the interface:

- Configuration entry `enableRTU = 0`

Information

Currently only Modbus TCP is supported. Modbus RTU must be deactivated.

Configuration of the Modbus TCP interface:

- Designation of the sub-node `[ModBus.TCP]`
- Configuration entry `IP`: IP address filter for server access
  `IP="0"` allows access from any client
  `IP="xxx.xxx.xxx.xxx"` allows only the access of the client with the entered IP address
- Configuration entry `PORT`: Port which the Modbus server opens
  `PORT = 502` is default port
- Configuration entry `TIMEOUT`: Timeout in s for monitoring a cyclic communication with an external client
  Currently not used.

Example configuration - interface

```
[ModBus]

    enableRTU = 0
    [ModBus.TCP]
    IP = "0" // each client is allowed    PORT = 502 // Default port
```

14.2.2 Configuration data

The following base nodes are used for the configuration of the data:

Name	Description
`[ModBusReg]`	Base node for holding register
`[ModBusCoil]`	Base node for coils
`[ModBusInputReg]`	Base node for input register (Read Only data)
`[ModBusInputCoil]`	Base node for input coils (Read Only data)

The data assignment is then carried out in sub-nodes.

The following parameters can be set for the configuration of registers and coils of the Modbus Server:

Parameter	Beschreibung
Modbus register / Coil address assignment	• The address is specified as an index at the corresponding node, e.g. `[ModBusReg.Adr:X]` • The numbering starts at 1 and runs up to a maximum of 65535 (0xFFFF). Gaps in the assignment can be configured.

Variable name	• Is specified as a sub-entry for the corresponding node e.g. `Variablenname = "APPL.Application.GVL.gint16_ro"`
	• The complete variable path must be specified (up to the root node "APPL")
	• Variables must have their access rights set in the "Symbol Configuration" of the u-create studio project so that they can be used for the Modbus server (Access Rights Read, Write, Write / Read)
	• The following variable types can be released for the Modbus server:
	  ◦ 8-bit data types: SINT, USINT, BYTE without restriction
	  ◦ 16-bit data types: INT, UINT, WORD without restriction
	  ◦ 32-bit data types: DINT, UDINT, DWORD, REAL: via double assignment for registers or via conversion factor and only INT number range
	  ◦ 64-bit data types: LREAL with conversion factor and cut to INT Number range
	Other variable types are not supported by the Modbus server and lead to error code when accessing the assigned register / coil `MODBUS_EXCEPTION_SLAVE_OR_SERVER_FAILURE`.

Write permission	• Is specified as a sub-entry for the corresponding node, e.g. `WritePermission = 0` • Value 0: read access only, 1:write and read access • For input registers and input coils the setting is ignored. Only read access by definition is possible here.

Parameter	Beschreibung
Conversion factor	• Is specified as a sub-entry for the corresponding node e.g. `Factor = 1.0` • The u-create studio application variable is multiplied by this factor during read access before the value is sent back via Modbus. • The value received via Modbus is divided by this factor before the u-create studio application variable is written for write accesses.

Example configuration - Basic

```
[ModBus]
    enableRTU = 0

    [ModBus.TCP]
    IP = "0"
    PORT = 502

//Configuration Holding Register
[ModBusReg]
//Example read access 16 bit variable
    [ModBusReg.Adr:1]
            Factor = 1.0
    Variablenname = "APPL.Application.GVL.gint16_rw"
        WritePermission = 1

//Configuration Input Register
[ModBusInputReg]
    [ModBusInputReg.Adr:1]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.gint16_ro"
    WritePermission = 0

//Configuration Coil
[ModBusCoil]
    [ModBusCoil.Adr:1]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.nfigugint8_rw"
        WritePermission = 0

//Configuration Input Coil
[ModBusInputCoil]
    [ModBusInputCoil.Adr:1]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.gint8_ro"
        WritePermission = 0
```

14.2.3 Extended configuration

Registers with variables that are based on data types with more than 16-bits must either be scaled via the conversion factor and then trimmed accordingly, or they must be split up into several registers.

For a division the variable name is entered in 2 register configurations. Access to variables that have been split into 2 registers (double registers) is only possible in a single read/write access via function codes 0x03, 0x04 / 0x10.

Example configuration - extended

```
[ModBus]
    enableRTU = 0
```

```
    [ModBus.TCP]
    IP = "0"
    PORT = 502

//Konfiguration Holding Register
[ModBusReg]
//Example write permission 16 bit variable
    [ModBusReg.Adr:1]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.gint16_ro"
        WritePermission = 0

//Example read access floating point variable
    [ModBusReg.Adr:2]
        Factor = 100
        Variablenname = "APPL.Application.GVL.greal_ro"
        WritePermission = 0

//Example read access 32 bit variable via double register
    [ModBusReg.Adr:3]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.gint32_ro_multiReg"
        WritePermission = 0
    [ModBusReg.Adr:4]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.gint32_ro_multiReg"
        WritePermission = 0

//Example read access floating point variable via double register
    [ModBusReg.Adr:5]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.greal_rw_multiReg"
        WritePermission = 1
    [ModBusReg.Adr:6]
        Factor = 1.0
        Variablenname = "APPL.Application.GVL.greal_rw_multiReg"
        WritePermission = 1
```

ransmission of a 32bit floating point value via scaling and single register assignment

Variable `APPL.Application.GVL.greal_ro` ist im Konfigurationsbeispiel auf Einzel-Register 2 zugeordnet und mit Umrechungsfaktor 100 parametriert. Wird Register 2 über Modbus gelesen, wird der Variablenwert vor der Übertragung mit dem Umrechungsfaktor multipliziert. Der Empfänger muss die Skalierung durch Division mit dem Umrechungsfaktor wieder rückgängig machen.

Example:

`APPL.Application.GVL.greal_ro = 23.456`

The value multiplied by a factor of 100 and converted to an integer value. 23.456 * 100 = 2345.6 --> Integer conversion (without rounding) --> 2345

The value 2345 is transmitted via Modbus

Transmission of a 32bit floating point value via multi-register access

In the configuration example, variable `APPL.Application.GVL.glreal_rw_multiReg` is divided into registers 5 and 6 and can be transmitted as a 32 bit value via this double register.

The variable can only be read via function code 0x03 (Read Holding Registers) and written via function code 0x10 (Write Multiple Registers). The number of registers to be read / written is thus 2 or more (if further registers are manipulated).

The receiver must take care to reassemble the two register values to a valid floating point value during read accesses.

Example:

```
APPL.Application.GVL.greal_rw_multiReg = 23.456
```

Display as 32 bit number in hex format: 0x41BBA752, split into 2 16bit registers: 0x41BB, 0xA752

Thus, register 5 contains the high word 0x41BB (= 16827), register 6 contains the low word 0xA752 (= 42834). These values are transmitted via Modbus.

To check the binary representation of the transmitted values, various tools can be used:

- HEX↔Dezimal converter
  http://www.binaryconvert.com/convert_unsigned_short.html
- FLOAT↔HEX converter
  http://www.binaryconvert.com/convert_float.html

14.2.4 Configuration of Arrays

Using a special syntax, data arrays of the controller can be easily configured for the Modbus server. For this purpose, the start index and the end index of the array are specified for the address assignment at the sub-node according to the IEC syntax.

Modbus Register / Coil Address Assignment:
The address is specified as index at the corresponding node followed by the start index and end index of the array e.g

```
[ModBusReg.<Modbus Addr>:<array start index>:<array end index>]
```

Example configuration - Arrays:

Definition of the array with 2000 elements in the program code of the control application.

```
VAR_GLOBAL
    dataArrayModbus : ARRAY [1..2000] OF REAL := [2000(0)];
END_VAR
```

Data assignment to the Modbus holding registers starting with register index 200:

```
[ModBusReg.Adr:200:1:2000] //test array configuration
    Factor = 1.0
    Variablenname = "APPL.Application.GVL.dataArrayModbus"
    WritePermission = 1
```

The array is thus assigned to the registers 200 to 2199.

15 Data recorder

Using the data recorder, variable values can be recorded on the control during runtime.

Access to the data recorder is possible via the following interfaces:

- Library "KREC" for IEC applications (real-time access) in u-create studio
- "DataRecApi" interface for C applications

For additional information, refer to the online help for u-create studio.

A data recorder has to be created and configured by the application first. This is possible via the IEC application or using u-create scope.

15.1 Configuration

Within an IEC application, one (or more) data recorders can be created and configured. The necessary configurations (properties, variables, etc.) are compiled and stored in what is known as a "profile".

The basic properties of a profile are the following:

- Maximum number of variables
- Buffer capacity
- Buffer type
- Persistence

Maximum number of variables

Every variable whose value is to be recorded must be registered with the data recorder. This requires the complete path of the variable. The maximum number of variables defines the upper limit for the number of variable registrations. Variables can be registered and deregistered, but the number cannot exceed the specified number of variables.

Only variables with the following data types can be registered with the data recorder:

BOOL	REAL	LREAL	BYTE
ENUM	WORD	DWORD	LWORD
SINT8	SINT16	SINT32	SINT64
ENUM_SINT8	ENUM_SINT16	ENUM_SINT32	ENUM_SINT64
UINT8	UINT16	UINT32	UINT64
ENUM_UINT8	ENUM_UINT16	ENUM_UINT32	ENUM_UINT64
DATE	DATE64us	TIME	TIME64us
DT	DT64us	TOD	TOD64us

Buffer capacity

The buffer capacity defines the maximum number of stored variable values with a time stamp that a profile can store. The recording process reads out the values of all registered variables and writes these into the buffer of the profile along with a time stamp. This data record (variable value with time stamp) is called "Sample".

The buffer capacity multiplied by the maximum number of variables determines the storage requirement of a profile. The memory must be created beforehand in order to ensure recording in real time.

Buffer type

The following types of buffers can be configured:

- **Continuous** (endless recording): Recording beyond the buffer capacity is possible by starting from the beginning once again after the last entry. It is possible that overwriting unread samples may result in the loss of data. When reading data, data should be checked for completeness.
- **SingleShot** (individual recording): no recording beyond the buffer capacity. At the latest, the recording ends when the end of the buffer is reached. No data loss, no verification of the completeness of read data necessary.

Persistence

By default, configurations and stored data are not kept after the system is restarted. However, this can be changed by configuring the data recorder.

The following configurations are possible:

- No persistence: Each time the application starts-up, the profile is newly created by the application and the recording starts with an empty buffer.
- Persistence of the configuration: The profile settings are stored in a file under the path `{System.applPath}/application/control/config`. In the course of the start-up of the control, the profile is automatically created according to the configuration. The recording starts with an empty buffer. The data is not persistently stored. If the `autoStart` option is selected when the profile is created for the first time, recording is automatically started during start-up.
- Full persistence: All information (configuration, as well as data that is already recorded) is persistently stored. The state and data are binary-stored in the directory `{System.applPath}/retain`. In the course of the start-up, the profile is automatically created according to the configuration, and is put into the state that it was it at the last point in time a save operation was carried out. If the `autoStart` option is selected, the recording continues.

With regard to the persistence, the following shall be observed:

- When the profile state is being stored, large amounts of data can accrue, depending on the size of the profile. If the application stores the state cyclically at frequent intervals, this can significantly reduce the service life of non-volatile storage. After the application controls the storage of the states themselves, a sensible compromise should be sought here.
- Changes to the application can cause persistent settings of a profile to become invalid if they reference variables that no longer exist. Invalid settings, such as faulty trigger variables, result in the profile no longer being able to be configured as it was originally. The application discovers this through corresponding error messages. If, however, there are only individual variables that are missing that have been registered for recording, the profile is configured with the available variables as being ready-to-use (and, if necessary, also started).
- If a profile is deleted, its persistent data is also deleted (configuration and recording status). If the queue is deleted, the persistent recording status is also deleted.

15.2 Data recording

Variables that are registered to a profile can be recorded. A recording consists of

- capturing the values of all registered variables and
- the storage thereof (together with metadata) in the buffer.

The recording of the variables can be controlled in terms of time or event. The data is stored in a buffer in conjunction with a time stamp. The buffer is located in the RAM, but it can also be stored on a data carrier. The buffer can be read out area by area, even during a running recording.

Recording can take place either manually or automatically. During automatic recording, the following can be configured by means of the application:

- Temporal distance of the individual recordings in µs
- Task priority of the individual recordings (from 1 = high to 31 = low)
- CPU task binding (starting from 0, or -1, for system-side selected standard binding, > 1 CPU available for the runtime)

With these settings, a task is generated on the system side, which is queued under the usual application tasks in regards to intermittency and priority.

Information

The interaction of these tasks with the usual application tasks must be taken into account. For example, if the time stamps of the data detect unwanted “jitter,” the adopted configuration should be modified. Furthermore, the priority should not be too high, the distance should not be too short and the quantity of data should not be too large in order to prevent a blocking of real-time-critical tasks.

A profile can occupy the following states:

State	Description
recording	A recording is carried out.
stopped	The recording is stopped or ended.
waiting	The start trigger or start delay is waited upon before the recording is carried out.

Recording mode

The following recording types are possible:

- Standard: the values of all variables are stored per individual recording
- Change-based: The values of all variables are stored upon a change of a certain variable

A change-based recording does not monitor all variables of the profile, but rather only a specific variable. If its value changes, the values of all profile variables is are recorded. Optionally, the recording of the values with respect to the detection of a change can be reduced in importance.

Example

Change-based recording with a monitored variable `prodCnt` and a reduction of the storage by a factor of 2, only the values that are written in bold are stored.

prod-Cnt	**1**	1	2	2	**3**	3	4	4	**5**
weight	**14**	15	16	14	**15**	13	14	15	**16**
tmp	**60**	60	60	61	**61**	60	60	60	**60**

Information

The initial sampling of a variable since its registration with a profile always results in it being stored, since the variable value is in any case applied as being changed.

Quantity limits

It is relevant for some applications that recordings should not run endlessly, but instead should be automatically stopped after a defined number of recordings. Furthermore, it may be sensible not to begin the recording immediately after start-up, but rather only after a certain time frame (especially if a trigger actuates the start and relevant data is only available a little while after the trigger).

For this purpose, the following limits can be set:

- Start delay: Defined number of calls after recording start, that do not generate any recordings. Only after this quantity are the variable values recorded.
- Number of the recordings to be stored: Number of calls (without start delay) until the recording is automatically stopped.

Start and stop trigger

Triggers are conditions, which can cause a recording of a profile to be started or stopped in an event-driven manner. A trigger condition always refers to a certain variable. This does not have to be registered in the profile, but rather is specified via its complete path in the variable tree. The same limits apply for the data type as for registered variables. The following types of conditions can be specified with these variables:

- Exceeding of a constant threshold value t: The condition occurs when the variable value t is exceeded, while upon the previous call, this was not yet the case.
- Falling short of a constant threshold value t: The condition occurs if the variable value t is fallen short of, while upon the previous call, this was not yet the case.
- Exceeding or falling short of a constant threshold value t: OR a combination of both of the above conditions
- Any change of the value of the variable: The condition occurs if the variable value has changed with respect to the last call

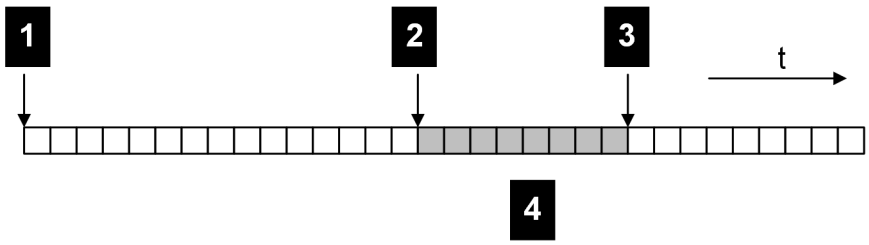
Upon the occurrence of the start trigger condition, the recording begins corresponding to the configuration (in other words, either endless or until the end of the buffer is reached). If a stop trigger condition has been configured, the recording is stopped upon the occurrence of this condition.

A start trigger condition can also be set without an associated stop trigger condition. The converse is not possible.

Furthermore, there is the option to define a stop time condition (post-start trigger). This defines a number of recordings. Afterwards, the recording is stopped automatically.

Example

Total capacity: 32 samples, stop time condition: 25 % (= 8 samples)

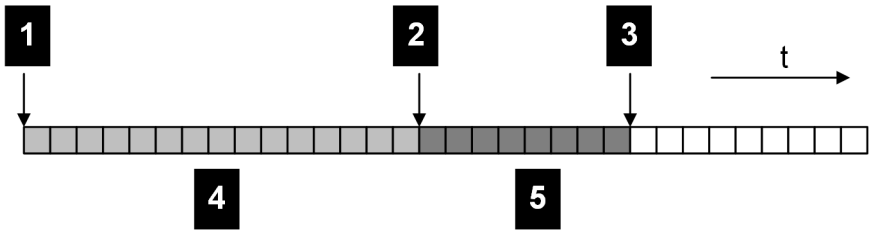


1 ... Recording start	2 ... Start trigger
3 ... Recording stop	4 ... Stop time

Information

For the assessment of the stop time capacity as a percentage, it shall be observed that in the case of the “continuous” buffer type, the buffer capacity is rounded up system-internally to the next power of two (e.g. 4000 is rounded up to 4096). The percentage has an effect on the value that is rounded up to.

In order to record data before the occurrence of the start trigger, a start time must be configured in the application (pre-start trigger recording). In this case, the buffer is filled immediately after the recording start. Upon the occurrence of the start trigger, the recording continues as part of the stop time condition, or the recording stops if a stop time condition has not been configured.



1 ... Recording start	2 ... Start trigger
3 ... Recording stop	4 ... Start time
5 ... Stop time	

Total capacity: 32 samples	Stop time: 25% = 8 samples, max. start time: 75% = 24 samples

Information

For start time and stop time conditions, the writing position is stored in the profile at the point in time the trigger condition occurred. These positions remain available until the next recording.

The start time and/or stop time condition overwrites the following configurations:

- The start time rescinds the start delay
- The start time or stop time rescinds the stop trigger
- The stop time rescinds the number limitation

Effectiveness of settings

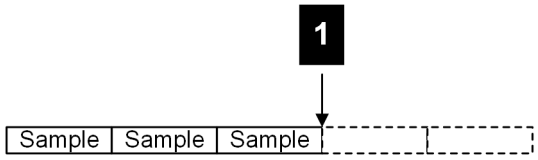
All settings carried out using set and remove functions always take effect upon the next time a recording is started. Ongoing recordings are not immediately influenced by set and remove calls.

Exceptions include the login and logoff of variables. These functions also have an effect while a recording is ongoing, thereby enabling a dynamic configuration of variables without interrupting value processes.

15.3 Reading out the buffer

You can read out the samples stored in the buffer of a profile at any time, even during an ongoing recording. Each profile carries a write position on which the next individual recording will be stored. This position can be prompted.

Example



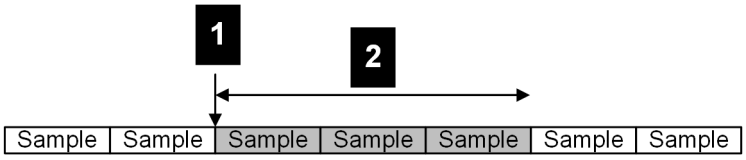
1 ... Write position	

When using triggers, the positions at which a call has detected the occurrence of the corresponding trigger condition are delivered.

When using a start and stop trigger, the difference between the delivered trigger positions is the number of the recorded samples minus 1, because the stop position designates the position of the sample that was recorded most recently.

The read functions enable the reading out of any buffer areas, whereby an area is indicated by start position and sample number.

Example



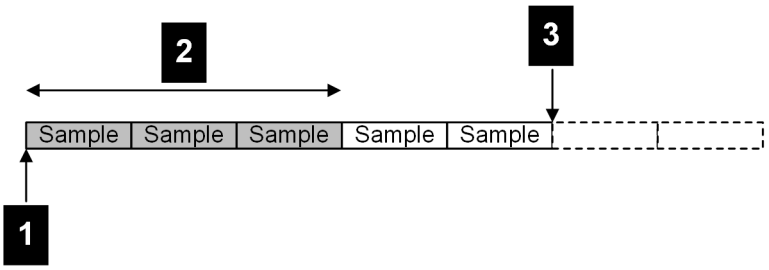
1 ... Start position	2 ... Number

In order to enable step-by-step reading, a position is also delivered in addition to data, which can be used as a start position upon the next call.

In addition to the delivered read positions, reserved positions for the oldest and newest samples are provided for first time reading.

Example

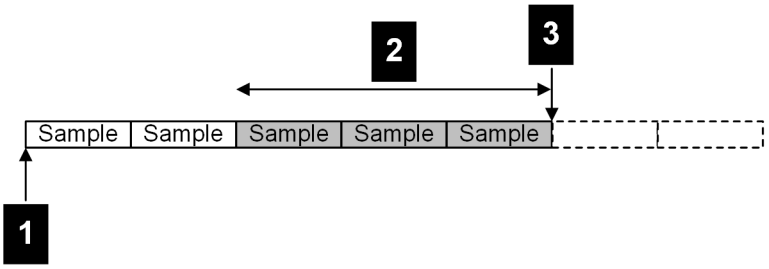
Start position: Oldest sample



1 ... Buffer start	2 ... Number
3 ... Write position	

Example

Start position: Newest sample



1 ... Buffer start	2 ... Number
3 ... Write position	

The validity of a reading position is limited in the context of the buffer type "Continuous". Upon reaching the end of the buffer, the write index breaks down, by which existing samples are overwritten. The read functions detect when a start position refers to samples which meanwhile have been overwritten. In this case the requested data cannot be delivered. The output parameter then contains correspondingly fewer values for actually delivered samples. The delivered reading position for subsequent calls refers to the current buffer state.

15.4 Recording callback from the application

For some applications, it may be sensible to combine the recording of process variables in data recorders with process-oriented testing and/or further processing. Over the course of each individual recording, a variable is read only one time, but the value can be used for several actions.

For this purpose, a data indicator can be registered, which is called up at the end of an individual recording and receives the result of the recording as arguments as well as the data indicator from the registration. The recording includes the following information:

- Time stamp of the sample
- Recording position

- List of the stored samples of variables, consisting of variable ID and variable value

All information is available as read only. They are stored in the profile buffer before the application function call. For one thing, data fidelity is thus ensured, and for another thing, the intermittency of the recording is independent of fluctuations that develop as a result of irregular run times of the application function.

Information

The application function should have the shortest and the most constant runtime possible. In the event that an automatic recording has been configured for the profile, it must not contain any blocking calls, since unlimited wait times have negative impacts on the recording of other profiles.

15.5 Data locality and real-time

The variable server concept, on which the data recording in profiles is based, is defined across processes. That means that variables are physically distributed to different system processes. The determinism of data recordings thus depends on the data locality: Access to process-local variables lasts shorter, while access to remote variables by means of interprocess communication lasts longer.

In order to achieve the highest possible determinism for real time-relevant recordings, process-local variables that are of a real-time quality should not be registered in the same profile together with remote variables. In lieu of this, it is recommended to separate profiles of a real-time quality – which may possibly also scan in shorter cycles and higher priority – from the remaining profiles.

15.6 Size limitations

In consequence of the current technical implementation for 32 bit systems (max. 4 GB virtual address space, max. 32 bit), the following theoretical limitations apply in terms of the amount of data:

Max. buffer capacity	Max. number of variables
524,288	1
262,144	2
131,072	4
65,536	8
32,768	16
16,384	32
8,192	64
4,096	128

Max. buffer capacity	Max. number of variables
2,048	256

Example

Profiles with between 32,769 and 65,536 samples can register a max. of 8 variables for recording.

Profile with SingleShot buffer type

Max. buffer capacity	Max. number of variables
1,048,576	254
1,042,453	256

Profiles with a maximum capacity of 1,048,576 can register a max. of 254 variables for recording. Conversely, the maximum of 256 variables can only be registered to profiles with a maximum capacity of 1,042,453. The maximum refers to 4 GB – for Linux with, for example, 1 GB of user address space, the information must be divided by four and, depending on the design of the memory, divided once again.

15.7 Appendix: C interface

The following section contains examples for the use of the `DataRecApi` programming interface for C applications.

All examples use the following configuration:

```
[Xcrt]
    traceWord = -1

[Xcrt.Resource:0]
    name = "resource7"

[Xcrt.Resource:0.Task:0]
    name = "Task_1s"
    interval = 1000000
    priority = 5

[Xcrt.Module:0]
    codeFile = "libTestDataRecApp"
    moduleInitFunc="moduleInit"
    moduleStartFunc="moduleStart"
    moduleStopFunc="moduleStop"
    moduleExitFunc="moduleExit"

    [Xcrt.Module:0.TaskConnection:0]
        context = "resource7.Task_1s"
        prio = 1
        hookClientFunc = "moduleCallback"
```

In addition, all examples share the same header:

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "TestDataRecApp.h"
#include "MemApi.h"
#include "LogApi.h"
#include "DataRecApi.h"
/
```

```
*****************************************************************************
* interface functions
*/
ModuleInitFunc moduleInit = &ModuleInit;
ModuleStartFunc moduleStart = &ModuleStart;
ModuleStopFunc moduleStop = &ModuleStop;
ModuleExitFunc moduleExit = &ModuleExit;
HookClientFunc moduleCallback = &ModuleCallback;
/
*****************************************************************************
* macros
*/
#define ReturnIfNot(cond) \
if (!(cond)) { LogApiTrace(traceid, "### ReturnIfNot@line %d ###",
__LINE__);return;}
/
*****************************************************************************
* constants
*/
static const char scProfileName[] = "TestDataRecAppP1";
static const char scVarNameTemp[] = "temperature";
static const char scVarNamePres[] = "pressure";
/
*****************************************************************************
* static variables
*/
static int32_t sTemp;
static float sPres;

static DataRecApiProfileId sP1Id;
static DataRecApiVarId sTempId;
static DataRecApiVarId sPresId;
```

General instructions

Automatic recordings are to be configured in C-applications with the `DataRecApiSetRecContext` function. The function defines a task context for the recording based on the following information:

- Temporal distance of the individual recordings in µs
- Task priority of the individual recordings (from 1 = high to 31 = low)
- CPU task binding (starting from 0, or -1, for system-side selected standard binding)

With these settings a task is generated on the system-side, which is queued under the usual application tasks in regards to intermittency and priority. To fulfill certain requirements on the determinism of the recording, the interaction of this task with the remaining application tasks is to be taken into account. For example, if the time stamps of the data detect unwanted “jitter,” the selection of the above parameters is to be reconsidered. Furthermore, the priority should not be too high, the distance should not be too short and the quantity of data should not be too large in order to prevent a blocking of real-time-critical tasks.

Information

If manual recording is selected for a profile, note that calls from `DataRecApiSampleValues` are not only necessary for data recording, but also for state transformations in the profile. For automatic recording, the calls and the state transformations associated with them are made by the configured task. This must be implemented in the application for recording manually.

For example, a call from `DataRecApiSampleValues` involves the following additional steps, which can cause a state change from `waiting` to `recording`:

- Testing of a configured start-trigger condition for their occurrence
- Testing for the expiration of a start delay set with `DataRecApiSetRecCount`

Furthermore, in the case of the SingleShot buffer type, `DataRecApiSampleValues` causes a state change from `recording` to `stopped` when the end of the buffer is reached.

The recording mode (standard or change-based) of a profile can be configured using the `DataRecApiSetRecMode` function. In the change-based mode, the tolerance value can then be changed using `DataRecApiSetVarTolerance` or enabled using `DataRecApiRemoveVarTolerance`.

Information

In the change-based modes, note that a profile only detects changes of variable values in the course of the call from `DataRecApiSampleValues`. Writing the application to the monitored variable alone does not cause any recordings!

Using the `DataRecApiSetRecCount` function, the following limitations can be set:

- Start delay: Number of calls by `DataRecApiSampleValues` after the start of recording, which do not generate any recordings, but instead implement a countdown for the recording.
- Number of the recordings to be stored: Number of calls by `DataRecApiSampleValues` (without start delay) until the automatic recording stop.

Special case in the change-based recording mode with a monitored variable: The mere number of calls from by `DataRecApiSampleValues` is not a part of the number of recordings to be stored, since not every call needs to be stored. Instead, only the actual recording calls of the function are those which count. In the remaining modes, by contrast, the pure calls to `DataRecApiSampleValues` do count.

You can set a trigger using `DataRecApiSetTrigger` and rescind it again using `DataRecApiRemoveTrigger`.

Special case in the change-based recording mode with a monitored variable: The mere number of calls from by `DataRecApiSampleValues` is not a part of the number of recordings to be stored, since not every call needs to be stored. Instead, only the actual recording calls of the function are those which count. In the remaining modes, by contrast, the pure calls to `DataRecApiSampleValues` do count.

In the course of the registration, the `DataRecApiAddVar` registration function determines the locality of a variable. This information is made available to the user by means of the `DataRecApiGetFirstVar`, `DataRecApiGetNextVar` and `DataRecApiGetVars` query functions and can be used for inspection purposes. The delivered `isLocal` flag indicates whether the variable is process-local, and as such, that it can be sampled real-time capable.

Settings carried out using `DataRecApiSetSampleHook` are not permanent (code addresses depend on start-up), but rather must be newly carried out in the course of every start-up of the application (before the recording start).

15.7.1 Profile with manual recording

The following example shows the easiest application: The application registers two variables with a variable server via `MemApi` and registers their paths at a profile with the Continuous buffer type and a capacity of 1000 samples an.

The recording is started in the start function, while it is stopped again in the stop function. The `ModuleCallback` application function is called up in one-second intervals and logs the variables by means of a call by `DataRecApiSampleValues`.

It is not necessary to explicitly delete the profile in the exit function, since all remaining profiles are deleted after all applications are exited. Persistent data is not affected by this deletion.

```
void ModuleInit() {
    DataRecApiInfo info;
    DataRecApiResult rc;
    char varPath[80];
    LogApiAddId(traceid, "TestRec");
    LogApiTrace(traceid, "TestDataRecApp initialized");
    MemApiInit();
    MemApiAddVar(scVarNameTemp, MemApiSInt32, &sTemp, sizeof sTemp);
    MemApiAddVar(scVarNamePres, MemApiReal, &sPres, sizeof sPres);
    sTemp = 0;
    sPres = 0.0;
    strcpy(info.name, scProfileName);
    info.size = 1000; /* space for 1000 samples */
    info.maxVarCnt = 5; /* max. 5 variables per sample */
    info.bufType = DataRecApiBufTypeContinuous; /* ring buffer type */
    info.level = DataRecApiLevelApplRt; /* (unsupported yet) */
    info.autoStart = 0; /* application starts recording */
    rc = DataRecApiCreateProfile(&info, &sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk && sP1Id != DataRecApiNoId);
    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNameTemp);
    rc = DataRecApiAddVar(sP1Id, varPath, 0, &sTempId);
    ReturnIfNot(rc == DataRecApiResultOk && sTempId != DataRecApiNoId);
    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNamePres);
    rc = DataRecApiAddVar(sP1Id, varPath, 0, &sPresId);
    ReturnIfNot(rc == DataRecApiResultOk && sPresId != DataRecApiNoId);
}

/
```

```
*****************************************************************************/
void ModuleExit() {
    MemApiRemoveVar(scVarNamePres);
    MemApiRemoveVar(scVarNameTemp);
    MemApiExit();
    LogApiTrace(traceid, "TestDataRecApp unloaded");
    LogApiRemoveId(traceid);
}

/
*****************************************************************************/
void ModuleStart() {
    DataRecApiResult rc;
    LogApiTrace(traceid, "Starting TestDataRecApp");
    rc = DataRecApiStartRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
}
/
*****************************************************************************/
void ModuleStop() {
    DataRecApiResult rc;
    LogApiTrace(traceid, "Stopping TestDataRecApp");
    rc = DataRecApiStopRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
}
/
*****************************************************************************/
void ModuleCallback(UserFuncParam *arg, uint32_t provArg) {
    DataRecApiResult rc;
    rc = DataRecApiSampleValues(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
    sTemp += 1;
    sPres += 0.1;
}
```

15.7.2 Profile with automatic recording

The following example builds upon the aforementioned by adding two points: For one thing, recording takes place automatically. For that to happen, a sampling in 10ms intervals is set with `DataRecApiSetRecContext`, while furthermore a start delay of 2 seconds and a recording time of 5 seconds is set with `DataRecApiSetRecCount`. For another, a callback from the recording into the application is configured for the purpose of minimum-maximum determination.

```
void ModuleInit() {
    DataRecApiInfo info;
    DataRecApiResult rc;
    char varPath[80];
    DataRecApiContext ctx;
    DataRecApiCount cnt;

    /* for committed lines see example "manual recording" */

    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNamePres);
    rc = DataRecApiAddVar(sP1Id, varPath, 0, &sPresId);
    ReturnIfNot(rc == DataRecApiResultOk && sPresId != DataRecApiNoId);

    ctx.intervalUs = 10000;
    ctx.priority = 16;
    ctx.affinity = -1;
    rc = DataRecApiSetRecContext(sP1Id, &ctx);
    ReturnIfNot(rc == DataRecApiResultOk);

    cnt.startDelayCnt = 200;
    cnt.durationCnt = 500;
    rc = DataRecApiSetRecCount(sP1Id, &cnt);
```

```
    ReturnIfNot(rc == DataRecApiResultOk);

    rc = DataRecApiSetSampleHook(sP1Id, UpdateMinMaxTemp, 0);
    ReturnIfNot(rc == DataRecApiResultOk);
}
```

Callback function:

```
static int32_t sMinTemp = INT32_MAX, sMaxTemp = INT32_MIN;
/
*****************************************************************************
*/
static void UpdateMinMaxTemp(
    uint64_t timeStampUs, /**< [in] time stamp (µs since the epoch) */
    DataRecApiPos pos, /**< [in] sample position */
    DataRecApiVarSample *pSamples, /**< [in] variable samples */
    int32_t sampleCnt, /**< [in] length of \em pSamples */
    void *arg /**< [in] user argument */
    ) {
    int i;

    for (i = 0; i < sampleCnt; ++i) {
        if (pSamples[i].var == sTempId) {
            int32_t curTemp = pSamples[i].value.valSINT32;
            if (curTemp < sMinTemp) {
                sMinTemp = curTemp;
            }
            if (curTemp > sMaxTemp) {
                sMaxTemp = curTemp;
            }
        }
        return;
    }
}
```

15.7.3 Profile with triggered recording

The following example uses manual recording, whereby the span of recording time is limited by a start trigger and a stop trigger. Both of the two trigger conditions refer to the `active` Boolean variable created by the application. The start of the recording should be actuated by a transition from FALSE to TRUE, while the stop should be actuated by a transition from TRUE to FALSE. These conditions can be specified using the undershoot and overshoot operators. To do so, the `active` variable must initially have the value FALSE.

To be able to specify the path of the variables in the variable tree, they must be registered, like the recorded variables, by means of `MemApi`. However, unlike the monitored variables of a change-based recording, it does not need to be registered in the profile.

If a person sets the `active` variable to TRUE, and then a few seconds later back to FALSE again (in the example, not a component part of the application code), within the time span, the variables `temperature` and `pressure` are recorded. After the recording has been stopped, the number of the samples results from the difference of the trigger positions + 1, since the stop position indicates the position of the last sample.

```
/
*****************************************************************************
* constants
*/
enum { cFALSE = 0, cTRUE = 1 };
```

```
static const char scProfileName[] = "TestDataRecAppP1";
static const char scVarNameTemp[] = "temperature";
static const char scVarNamePres[] = "pressure";
static const char scVarNameAct[] = "active";

/
*****************************************************************************
* static variables
*/
static int32_t sTemp;
static float sPres;
static int8_t sActive;
static DataRecApiProfileId sP1Id;
static DataRecApiVarId sTempId;
static DataRecApiVarId sPresId;

/
*****************************************************************************/
void ModuleInit() {
    DataRecApiInfo info;
    DataRecApiTrigger trig;
    DataRecApiResult rc;
    char varPath[80];

    LogApiAddId(traceid, "TestRec");
    LogApiTrace(traceid, "TestDataRecApp initialized");

    MemApiInit();
    MemApiAddVar(scVarNameTemp, MemApiSInt32, &sTemp, sizeof sTemp);
    MemApiAddVar(scVarNamePres, MemApiReal, &sPres, sizeof sPres);
    MemApiAddVar(scVarNameAct, MemApiBool, &sActive, sizeof sActive);

    sTemp = 0;
    sPres = 0.0;
    sActive = cFALSE;

    strcpy(info.name, scProfileName);
    info.size = 1000; /* space for 1000 samples */
    info.maxVarCnt = 5; /* max. 5 variables per sample */
    info.bufType = DataRecApiBufTypeContinuous; /* ring buffer type */
    info.level = DataRecApiLevelApplRt; /* (unsupported yet) */
    info.autoStart = 0; /* application start recording */

    rc = DataRecApiCreateProfile(&info, &sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk && sP1Id != DataRecApiNoId);

    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNameTemp);
    rc = DataRecApiAddVar(sP1Id, varPath, 0, &sTempId);
    ReturnIfNot(rc == DataRecApiResultOk && sTempId != DataRecApiNoId);

    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNamePres);
    rc = DataRecApiAddVar(sP1Id, varPath, 0, &sPresId);
    ReturnIfNot(rc == DataRecApiResultOk && sPresId != DataRecApiNoId);

    memset(&trig, 0, sizeof trig);
    snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNameAct);
    trig.startCond.op = DataRecApiTriggerOpPosSlope;
    trig.startCond.val.valBOOL = cFALSE;
    strncpy(trig.startCond.var, varPath, sizeof trig.startCond.var - 1);
    trig.stopCond.op = DataRecApiTriggerOpNegSlope;
    trig.stopCond.val.valBOOL = cTRUE;
    strncpy(trig.stopCond.var, varPath, sizeof trig.stopCond.var - 1);

    rc = DataRecApiSetTrigger(sP1Id, &trig);
    ReturnIfNot(rc == DataRecApiResultOk);

}

/
*****************************************************************************/
void ModuleExit() {
    MemApiRemoveVar(scVarNameAct);
    MemApiRemoveVar(scVarNamePres);
```

```
    MemApiRemoveVar(scVarNameTemp);
    MemApiExit();

    LogApiTrace(traceid, "TestDataRecApp unloaded");
    LogApiRemoveId(traceid);
}

/
*****************************************************************************/
void ModuleStart() {
    DataRecApiResult rc;
    LogApiTrace(traceid, "Starting TestDataRecApp");
    rc = DataRecApiStartRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
}

/
*****************************************************************************/
void ModuleStop() {
    DataRecApiResult rc;
    LogApiTrace(traceid, "Stopping TestDataRecApp");
    rc = DataRecApiStopRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
}

/
*****************************************************************************/
void ModuleCallback(UserFuncParam *arg, uint32_t provArg) {
    DataRecApiSampleValues(sP1Id);
    sTemp += 1;
sPres += 0.1;
}
```

15.7.4 Profile with persistence

The following example is more strongly distinguished from the previous examples. It uses a profile with persistence and, as such, must cover a number of cases: For one thing, the profile does not yet exist and must be newly created, and for another thing, the profile is already created on the system side due to the existing persistence, and thus must only be restarted.

The recovery of profiles takes place between the init- and start calls of the application, because the paths of the configured variables are also persisted and these are only valid after the initialization of all runtime systems. This requires a change of the application configuration: The determination or the initial creation of the profile thus takes place in the course of the start call. This step is integrated into the extra function `InitRecording`. The remaining steps are also highlighted in separate functions in order to better uncouple the recording from the rest of the application code.

Likewise, in the course of the initialization, an application task is created, which completely persists the profile every 10 seconds. The `StartRecording` function triggers the start of the task, while the `StopRecording` function enables the task to be terminated.

```
static void InitRecording() {
    DataRecApiInfo info;
    DataRecApiResult rc;
    char varPath[80];

    /*
     * create new profile or get restored profile
     */
    sP1Id = DataRecApiGetProfile(scProfileName, &info);
```

```
    if (sP1Id == DataRecApiNoId) { /* create profile */
        strcpy(info.name, scProfileName);
        info.size = 1000;
        info.maxVarCnt = 5;
        info.bufType = DataRecApiBufTypeContinuous;
        info.level = DataRecApiLevelApplRt;
        info.autoStart = 0;
        rc = DataRecApiCreateProfile(&info, &sP1Id);
        ReturnIfNot(rc == DataRecApiResultOk && sP1Id != DataRecApiNoId);

        snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNameTemp);
        rc = DataRecApiAddVar(sP1Id, varPath, 0, &sTempId);
        ReturnIfNot(rc == DataRecApiResultOk && sTempId != DataRecApiNoId);

        snprintf(varPath, sizeof varPath, "APPL.MEM.%s", scVarNamePres);
        rc = DataRecApiAddVar(sP1Id, varPath, 0, &sPresId);
        ReturnIfNot(rc == DataRecApiResultOk && sPresId != DataRecApiNoId);

        {
            DataRecApiContext ctx;
            ctx.intervalUs = 2000000;
            ctx.priority = 16;
            ctx.affinity = -1;
            rc = DataRecApiSetRecContext(sP1Id, &ctx);
            ReturnIfNot(rc == DataRecApiResultOk);
        }
    } else { /* profile restored */
        rc = DataRecApiStopRecording(sP1Id);
    }
    sRun = 1;
    sStarted = 0;
    sTerminated = 0;
    sTaskHdl = CreateTask("LowPriorTask", LowPriorTask, 0, 24, 4096);
    ReturnIfNot(sTaskHdl != 0);
}

/
*****************************************************************************/
static void StartRecording() {
    DataRecApiResult rc;
    int ok;

    rc = DataRecApiStartRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);

    ok = ResumeTask(sTaskHdl); ReturnIfNot(ok);
    sStarted = 1;
}

/
*****************************************************************************/
static void StopRecording() {
    DataRecApiResult rc;
    DataRecApiPos pos;
    sRun = 0; /* tell LowPriorTask to terminate */

    rc = DataRecApiStopRecording(sP1Id);
    ReturnIfNot(rc == DataRecApiResultOk);
}

/
*****************************************************************************/
static void ExitRecording() {
    if (sStarted) {
        while (!sTerminated) {
            LogApiTrace(traceid, "waiting for LowPriorTask to terminate...");
            sleep(1/*secs*/);
        }
    }
}

/
*****************************************************************************/
```

```
void ModuleInit() {
    /*
     * create variables to be recorded
    */
    MemApiInit();
    MemApiAddVar(scVarNameTemp, MemApiSInt32, &sTemp, sizeof sTemp);
    MemApiAddVar(scVarNamePres, MemApiReal, &sPres, sizeof sPres);
}

/
*****************************************************************************/
void ModuleExit() {
    ExitRecording();
    MemApiRemoveVar(scVarNamePres);
    MemApiRemoveVar(scVarNameTemp);
    MemApiExit();
}

/
*****************************************************************************/
void ModuleStart() {
    /*
     * now all variables are available such that profiles can be restored and
     * variables can be added to profiles
     */
    InitRecording();
    sTemp = 0;
    sPres = 0.0;
    StartRecording();
}

/
*****************************************************************************/
void ModuleStop() {
    StopRecording();
}
```

Implementation of the persistence in 10 second intervals (with additional termination assessment per second):

```
/* libk2ctrl exports for advanced applications */
extern int CreateTask(const char *name, void (*fctAddr)(void *), void *param,
        int prior, int stackSize);
extern int ResumeTask(int hdl);

/
*****************************************************************************
* static variables
*/
static int sTaskHdl;
static int sRun, sStarted, sTerminated;

/
*****************************************************************************/
void LowPriorTask(void *arg) {
    enum { cSaveStatePeriodSecs = 10 };
    int cnt = cSaveStatePeriodSecs;
    DataRecApiResult rc;
    while (sRun) {
        sleep(1/*secs*/);
        if (--cnt == 0) {
            rc = DataRecApiSaveState(sP1Id);
            cnt = cSaveStatePeriodSecs;
        }
    }
    sTerminated = 1;
}
```

Information

Upon the creation of free-running tasks by means of `CreateTask`, the application takes over co-responsibility for the stability of the system.

*After the recurrence of the exit function (end of the application), make sure that all created tasks have been terminated or these **no longer** execute calls of the offered APIs (among others, `DataRecApi`)!*

15.7.5 Evaluation of recordings

The following code supplements the preceding examples by the evaluation of recordings that have been carried out. The parameters of the `PrintSamples` example function specify the buffer area to be read in the vertical (`pos`, `cnt`) and horizontal direction (`maxVarCnt`).

```
static void PrintSamples(int maxVarCnt, DataRecApiPos pos, int cnt) {
    char *pBuf;
    DataRecApiSample *pSample;
    DataRecApiVarSample *p;
    int actVarCnt, i, k, rc;

    pBuf = malloc(DataRecApiUtilGetSampleMemSize(maxVarCnt, cnt));
    ReturnIfNot(pBuf != 0);

    pSample = (DataRecApiSample *)pBuf;
    rc = DataRecApiReadSamples(sP1Id, maxVarCnt, &pos, pSample, &cnt);
    ReturnIfNot(rc == DataRecApiResultOk);

    for (i = 0; i < cnt; ++i) {
        actVarCnt = pSample->header.varSampleCnt;
        printf("[%d] recNo=%d, varSampleCnt=%d, timeStampUs=[%s]:\n",
        i,
        pSample->header.sampleRecNo,
        actVarCnt,
        TimeStamp2DateStr(pSample->timeStampUs));

    for (k = 0, p = pSample->varSamples; k < actVarCnt; ++k, ++p) {
        switch (p->type) {
            case DataRecApiVarType_SINT32:
             printf(" name=%s, value=%d (size=%d)\n", VarIdToName(p->var), p-
>value.valSINT32, p->size);
             break;
            case DataRecApiVarType_REAL:
             printf(" name=%s, value=%f (size=%d)\n", VarIdToName(p->var), p-
>value.valREAL, p->size);
             break;
            default:
             printf(" name=%s, value=%" PRIx64 " (type=%d, size=%d)\n",
VarIdToName(p->var), p->value.valNONE, p->type, p->size);
             break;
        }
    }
    pSample = (DataRecApiSample *)&pSample->varSamples[maxVarCnt];
    }
    free(pBuf);
}
```

Example call

```
PrintSamples(5, DataRecApiPosNewest, 1000);
```

Auxiliary functions for the formatting of time stamps and simple determination of variable names:

```
static const char *TimeStamp2DateStr(uint64_t timeStamp) {
static char sDateStr[80];
    time_t secs, usecs;
    struct tm *p;

    /* seconds and µs since 1970-01-01 */
    secs = (int)(timeStamp / 1000000);
    usecs = (int)(timeStamp % 1000000);

    p = localtime(&secs);
    if (p != 0) {
        snprintf(sDateStr, sizeof sDateStr, "%04u-%02u-%02u %02u:%02u:%02u.
%03u", p->tm_year+1900, p->tm_mon+1, p->tm_mday, p->tm_hour, p->tm_min, p-
>tm_sec, (unsigned)usecs);
    }
    else {
        snprintf(sDateStr, sizeof sDateStr, "xxxx-xx-xx xx:xx:xx.xxx");
    }
    return sDateStr;
}

/
*****************************************************************************/
static const char *VarIdToName(DataRecApiVarId varId) {
    if (varId == sTempId) {
        return scVarNameTemp;
    }
    if (varId == sPresId) {
        return scVarNamePres;
    }
    return "unknown";
}
```

16 Diagnostics

This chapter describes the diagnostic options of u-create. Depending on the delivery, not all diagnostics options may be available.

16.1 Control diagnosis

Errors during operation or operating states are indicated via the LEDs on the CPU module.

PWR (Power)

Indication	Meaning
Dark	No voltage supply
Green	Device running

SF (System Fault)

Indication	Meaning
Dark	No voltage supply or no error
Red	Severe system error (e.g. Fatal Error)

RUN

Indication	Meaning
Dark	No voltage supply
Green	Application is processed or setup finished
Flashing green	Application is stopped
Yellow	Ready for operation
Flashing yellow	Setup is executed
Flashing red	Setup failed

Link/Activity LED

Indication	Meaning
Dark	No connections
Flashing green/yellow	Data is transferred
Green	EtherCAT connection established (100 MBit/s, Full Duplex)

Via the Service App "DevAdmin" information of the control can be read and a state report as well as a crash report can be triggered. (see Device Administration (DevAdmin)).

16.2 Diagnosis data for Weidmüller

If you encounter a problem with the u-create system and require support from Weidmüller, please collect the following information from your system and send it to Weidmüller.

16.2.1 Status report

In order to retrieve the status data of the control, please use the status report function. It can be triggered as follows:

- Via Service-App (DevAdmin)

Triggering the status report via Service-App

See "Device Administration (DevAdmin)".

16.2.2 Crashreport

If a system error (e.g. "unhandled exception") occurs, a Crashreport file is created automatically and stored on the storage media of the control. The next time the control is started it will check if a crashreport file is available. If it is, relevant messages will be set in the message system. Generally, messages must be confirmed. The crashreport file can be stored via Service-App (see Device Administration (DevAdmin)) to an USB stick. For more information see the project engineering manual of the respective control.

16.2.3 Access to flash storage medium

Access to the flash memory medium can only be made via an SSH-encrypted SFTP or SCP connection. This requires a SFTP program installed on the PC. FTP programs can be downloaded from the Internet.

To establish a connection to the control using the SFTP program, the IP address of the control with the port number must be entered in the SFTP program. In addition, the following authentication data is required to establish a connection, which must also be entered in the SFTP program:

- Registration data
- SSH key

A detailed description can be found in the help of the SFTP program used.

Information

The selected SFTP program must support authentication using SSH encryption.

Registration data

The following users are created by default on the control and can be used to establish a connection:

- service
- admin

Information

The user data cannot be changed.

SSH key

For security reasons, it is necessary to change the SSH key pair (private and public key).

Changing the SSH key pair

A special program is required to generate a new SSH key pair. This can be downloaded from the Internet, where the program "PuTTYgen" (in Version < 0.75) is recommended.

User "service"

To change a new SSH key using "PuTTYgen", proceed as follows:

1) Starting "PuTTYgen" on the PC.

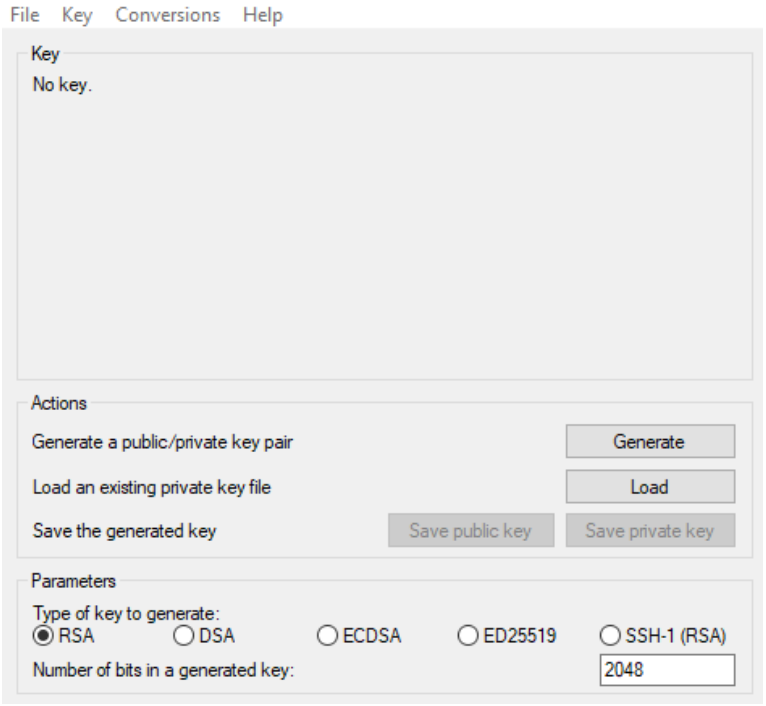
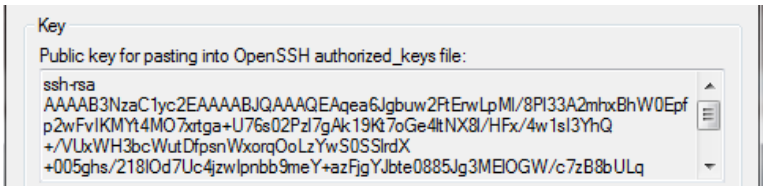


Fig. 16-19: PuTTYgen

2) Selection of the type "RSA" (selected by default) in the lower area of the dialog.
3) Generate the keys using "Generate".
4) Save the private key with "Save private key" under `[file-name]_key.ppk` into a directory on the PC.
5) Create a new text file `authorized_key.ppk` in a directory on the PC.
6) Select and copy the entire text in the text field at the top of the dialog under "Public key for pasting into OpenSSH authorized_keys file:".



7) Copy the text into the file `authorized_key.ppk`.
8) Save and close the file.

The SSH key pair has been generated. The public key must then be transferred to the control.

Proceed as follows to transfer the public key for the "service" user:

1) Start the SFTP program and connect to the control.
2) Change to the directory `ssh-key` and download the file `authorized_key.ppk` from the PC to the control (existing file will be overwritten).
3) Restart the control.

The SSH key has been changed. For a new SFTP connection to the control, the newly created private key must be specified.

User "admin"

To change a new SSH key using PuTTYgen, proceed as follows:

1) Starting "PuTTYgen" on the PC.

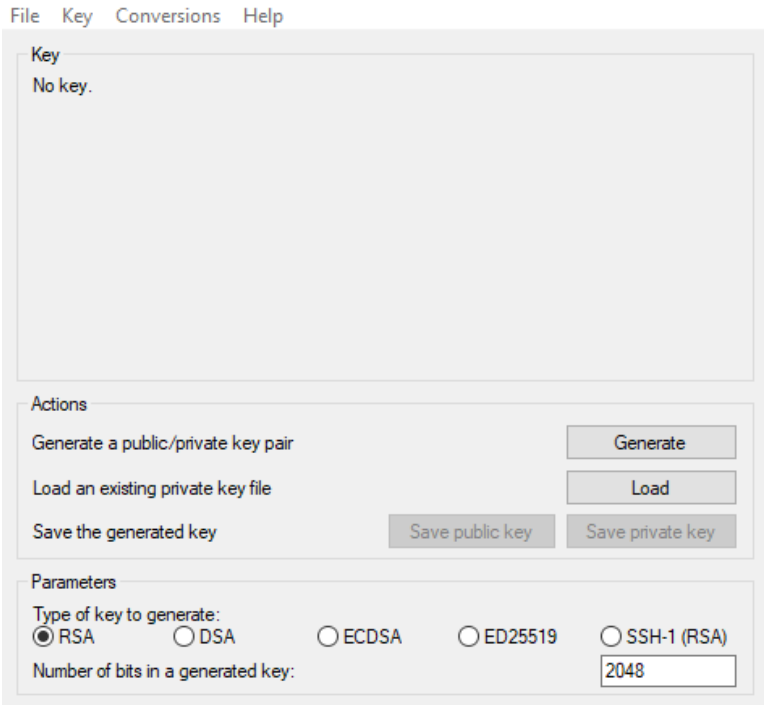
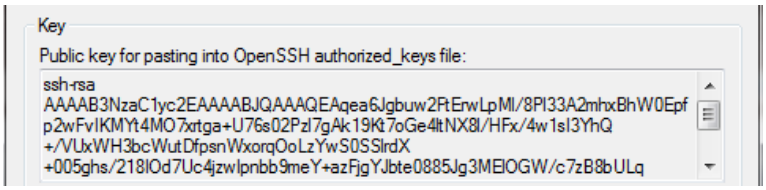


Fig. 16-20: PuTTYgen

2) Select the type "RSA" (selected by default) in the lower part of the dialog.

3) Generate the keys using "Generate".
4) Save the private key with "Save private key" under `[file-name]_key.ppk` into a directory on the PC.
5) Create a new text file `admin` in a directory on the PC.
6) Select and copy the entire text in the text field at the top of the dialog under "Public key for pasting into OpenSSH authorized_keys file:".



7) Copy the text to the file `admin`.
8) Save and close the file.

The SSH key pair has been generated. The public key must then be transferred to the control.

Proceed as follows to transfer the public key for the "admin" user:

1) Right mouse click on the "Software Service" icon in the task bar of the PC.
2) Open the directory with "Open Software Service Path".
3) Copy the file `admin` into the following directory (existing file can be replaced):

`SystemSoftware\u-create studio_[version]\dsarmxilinxkebwm-[version]_arm\config\key`

The new public key is now stored and will be transferred automatically when "Create Target" (via u-create studio) is executed the next time. This changed the SSH key. For a new SFTP connection to the control, the newly created private key must be specified.

16.3 USB via Ethernet adapter

The necessary driver to use Ethernet via USB can be found on the Weidmüller website at: www.weidmueller.com.

Information

The USB interface is automatically assigned the IP address 192.168.227.1.

17 Maintenance

The following items fall under the maintenance of the system:

- Executing a system backup or restore
- Executing a firmware update
- Information about potential errors and their rectification

The backup of the system data includes the following information:

- Firmware that is saved on the storage medium
- Application programs and files including configuration files, message texts
- Operating data that was saved by the application programs
- Data on SRAM and EEPROM (e.g. retain data, user data)

Information

It is recommended to generate a backup even on a new system as well as prior to any changes in the system (e.g. firmware update) and to store it securely in case a restore becomes necessary at a later point in time.

The execution of the maintenance measures is described in the respective project engineering manuals.

17.1 Executing a firmware update of the system components

The u-create system permits the central, automatic update of the software running on the hardware modules.

There are the following variants of updates:

- Run update automatically when the control is started up

Before the firmware update can be implemented, a removable disk (e.g. SD card) with the relevant data must be created.

Preparation

The following is required for the preparation of the firmware update:

- An empty removable disk with min. 2 GB.
- Run-compatible system on the control (must not have the latest version status).
- The required storage medium must be plugged into the PC.
- Firmware update tool installed on the PC.

The firmware update tool provides the following options:

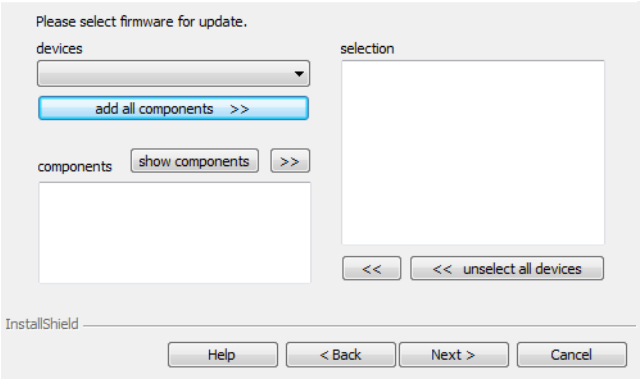
- Firmware update for control and bus coupler

17.1.1 Firmware-Update for devices

This chapter describes the creation of a storage medium to execute a firmware update of the control.

To create a firmware update on a storage medium proceed as follows:

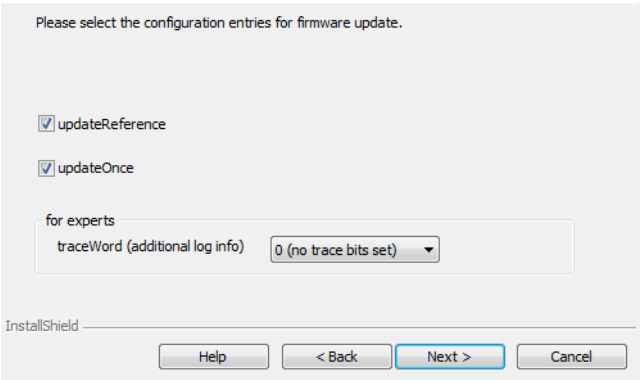
1) Start firmware update tool via double-clicking "u-create FirmwareUpdate.exe".
2) Confirm the following dialogue with "Next".
3) Select the desired device from the list "devices".
   Press "show components" to show all software components, for which an update is available, in the list "components".
4) Select the desired update and transfer it to the list "selection" via >>.



5) Remove one or several updates from the list "selection" via << or "<< unselect all devices".
6) To create updates for several devices, repeat step 4 and 5.

Bestätigen mit "Next".

7) Select the root directory of the connected storage medium and confirm with "Next".
8) The following configuration dialogue opens:



updateReference	... The update data will be additionally stored on the device.
updateOnce	... The update can only be executed once. It is not possible to use the removable media on another device.

traceWord	... 0: the execution of the firmware update will not be recorded. -1: the execution of the firmware update will be recorded in the system trace.

9) Click on "Next".
10) The storage medium will be created via "Install".

Information

The storage medium must not be removed while it is being written to!

11) Exit the firmware update tool via "Finish".

Information

It is recommended to execute "Eject device" under Windows before removing the storage medium.

The sotrage medium for updating was created and can now be removed from the PC.

17.1.2 Automatic update during boot-up

Proceed as follows:

1) Switch off the control.
2) Connect the SD card to the control.
3) Switch on the control.
4) During the update, the control is in the INIT status. After a while, the RUN LED changes from orange to flashing green.

Information

During the update, the power supply to the control must not be disconnected.

5) Open the handle and remove the SD card.
6) Restart the control.

The firmware and/or the bootloader is updated.

17.2 Installing a hotfix

A hotfix is used for the subsequent, short-term correction of local errors in a system component outside the regular release cycle. A hotfix is provided in the form of an additional software unit and installs itself to a defined, compatible system version. The hotfix is represented in u-create studio under Software Service as its own system version. Once a software unit containing a hotfix has been installed, it is taken into account when creating a target or transferring an application online to the controller.

Install hotfix

1) Run the `Setup.exe` file with administrator privileges.
2) Click **Next**. The following information is displayed:

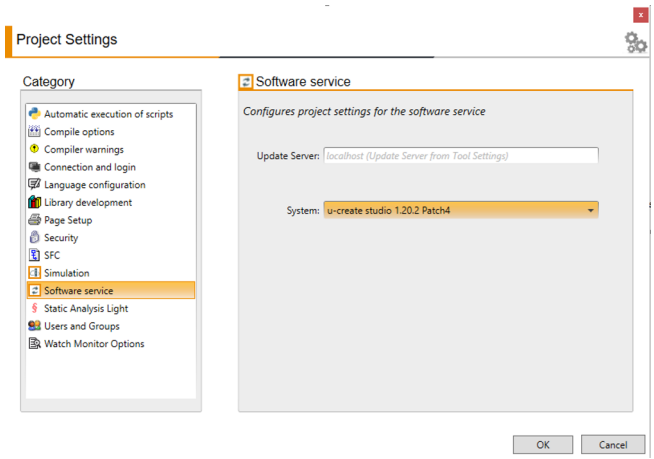
Software Unit Name	Name of the software unit to be installed
System Version Name	Name of the resulting system version
System Base Version	System version base on which the hotfix is applied

3) Click **Next** to start the installation.
4) Click **Finish**.

The hotfix has been installed.

Use Hotfix

1) Start u-create studio.
2) Select the installed hotfix under **Tools ► Options ► Software service**.



3) Select the installed hotfix (System Version Name from Setup dialog) under **System**.
4) Click **OK**.

The installed hotfix is used.

Once the hotfix is set in u-create studio, it can be applied to the machine via "Create Target" (offline) or "Software Update" (online).

Information

- The software version of the machine is changed by a hotfix. If an application is transferred to the machine, the changed software version is retained by selecting "Application only".
- A hotfix can be uninstalled at any time without affecting the original system version.

18 Technical data

Detailed information concerning the technical data can be found in the respective manuals.

19 EU directives, standards and regulations

Detailed information about directives and standards can be found in the project engineering manual of your modules.

20 Appendix: Tutorial - creating an IEC project

This tutorial describes step-by-step how a u-create studio project is created and programmed. Then, it illustrates how the project is uploaded onto a control and executed.

20.1 Creating a new project

Start u-create studio and call the command **File > New project** the main menu.

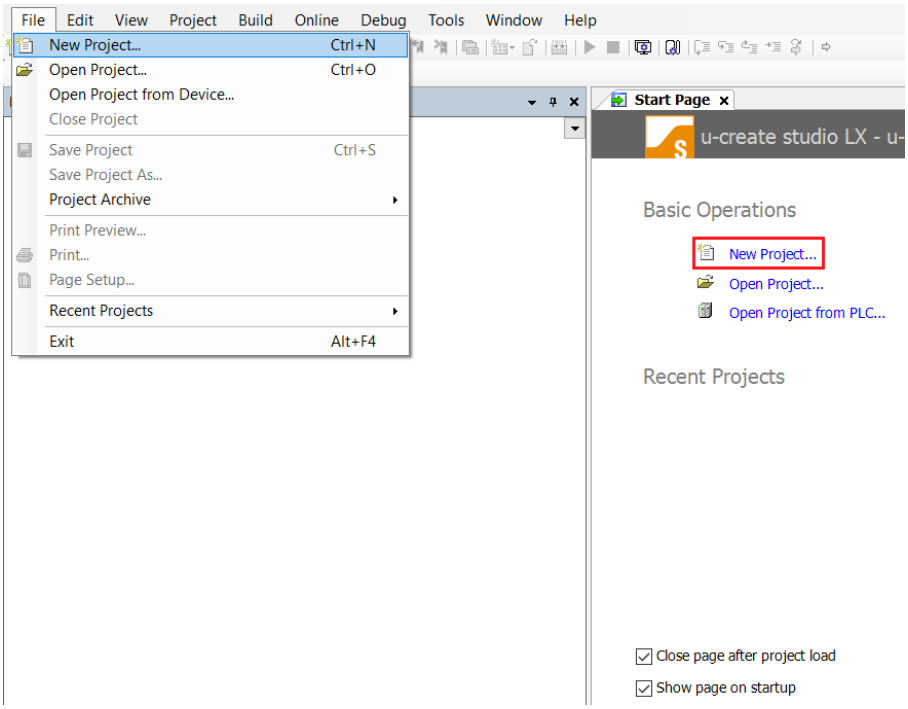


Fig. 20-21: u-create studio - New project

In the "New Project" dialog, select an empty project and enter a name and a location to save the project:

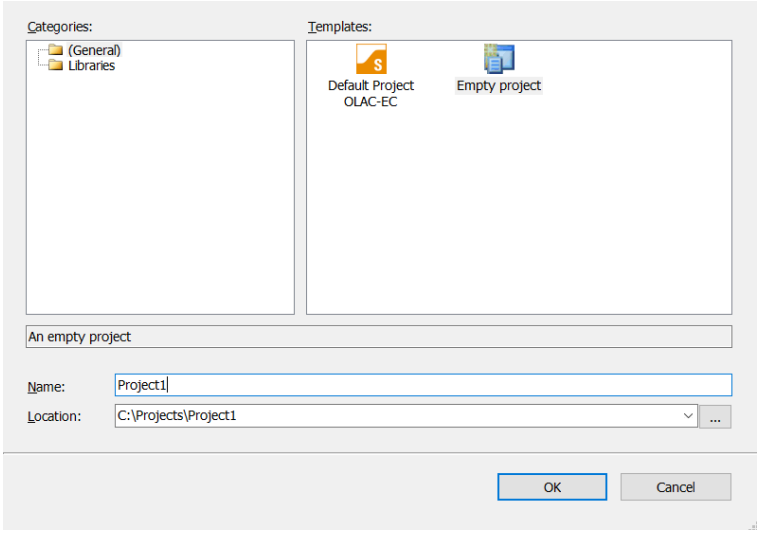


Fig. 20-22: Creating a new project

u-create studio automatically creates the project in the indicated directory and issues the addition "`.project`".

The empty project opens and a target system can be inserted into the project tree. To do this, open the context menu of the project in the project tree and click on "Append Device...". This opens a dialog to select a control. The desired control is highlighted in this dialog and added to the project via "Append Device" or by double-clicking on it. After that, the dialog can be closed.

A program module can then be created in the project tree under the **<Project name> > SPS logic > Application** node. To do this, open the context menu for the node **Application** and select **Add object > POU...**.

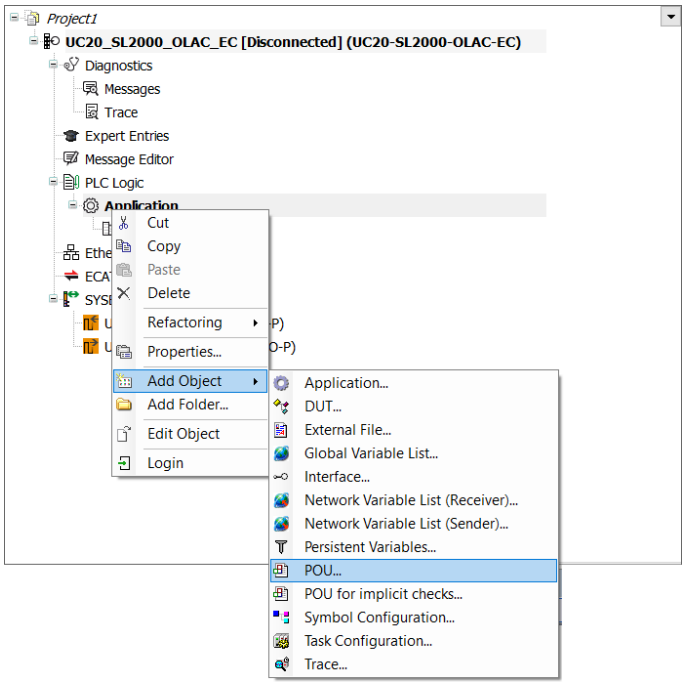


Fig. 20-23: Add POU

A dialog opens where you can set the name, the type, and the implementation language of the POU. In this example, the name "myProg", the type "Program", and the language ST (Structured Text) are used for the block:

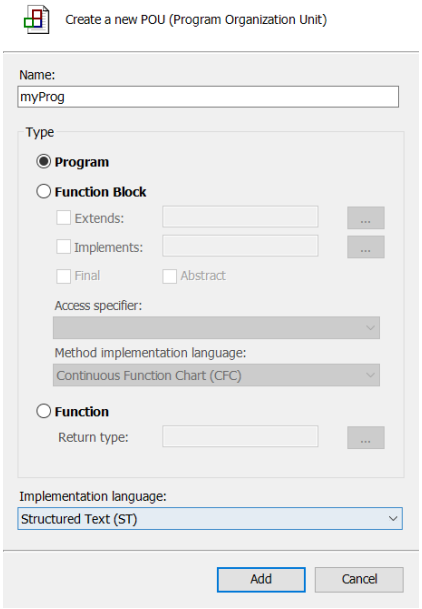


Fig. 20-24: POU settings

After the settings have been selected, the POU is inserted under the node **Application** via "Add"; the program editor automatically opens and the work area is ready for additional programming.

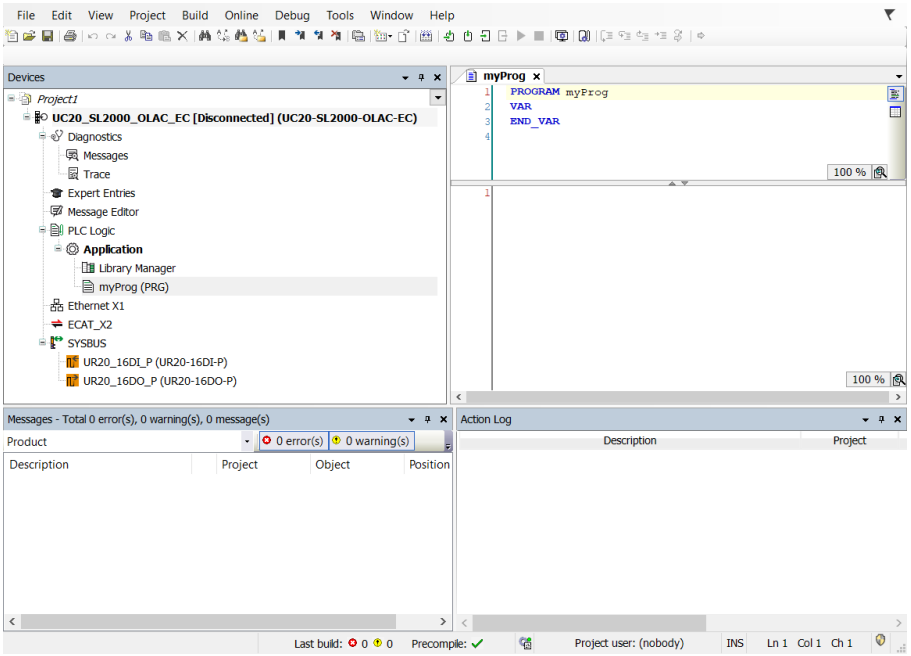


Fig. 20-25: Program editor of the POU

20.2 Creating a simple program

The newly created POU is now expanded through programming: A numerical variable needs to be created that changes its value cyclically.

The necessary entries (declaration of a numerical variable, value change of the variable) can be carried out in the displayed editor via keyboard entries in the programming language ST (structured text) selected during the creation of the module. The upper editing area is intended for variable declarations and the editing area underneath (still empty) for program actions.

However, there are also tools in u-create studio that help to create the correct program commands. Dialogue supported variables can be created via the main menu command **Edit > Auto Declare...**:

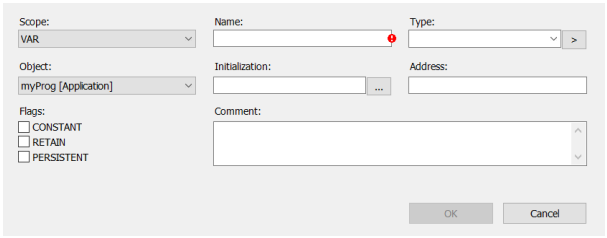


Fig. 20-26: The dialogue "Variable Declaration" supports the creation of a variable.

Enter the name and data type of the variable. If you are not yet familiar with the data types available in u-create studio, click on the button ">" next to the entry field "Type" and via "Input assistance" open an input Wizard to select the desired data type there:

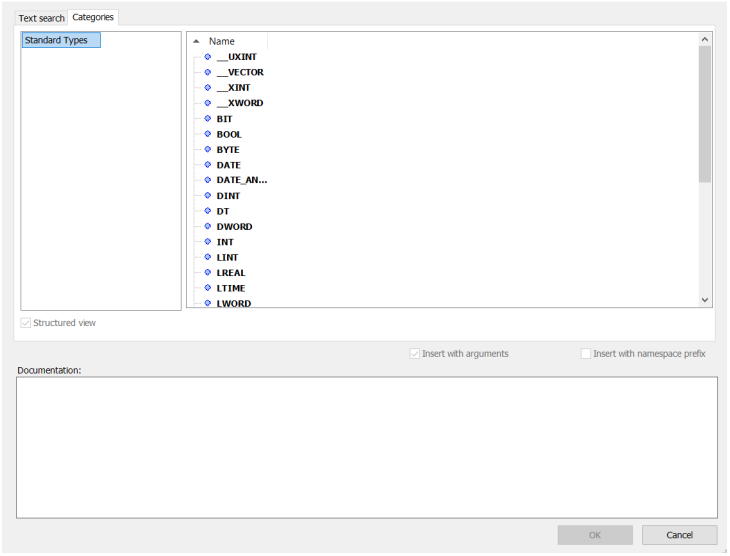


Fig. 20-27: Input assistance for the selection of a data type

In this tutorial, a variable X of the type DINT is to be declared:

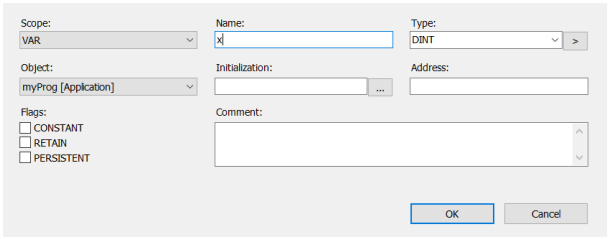


Fig. 20-28: Dialog "Variable Declaration" with entered variable name and data type

The created command (declaration of a variable) is inserted in the top part of the editor. The command for this cyclical value change of this variable must be inserted manually in the programming area of the editor below:

```
x := x + 1;
```

The finished code in the u-create studio editor looks as follows:

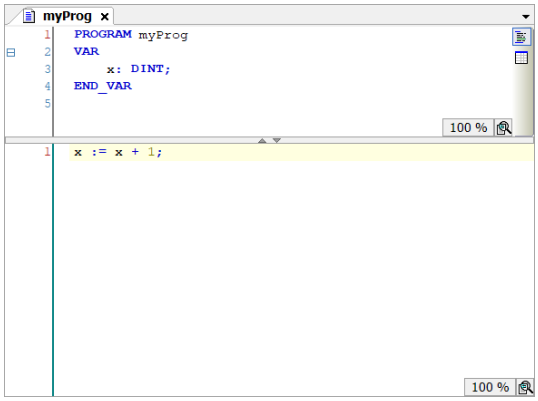


Fig. 20-29: Simple program in u-create studio

After creating the program, a task must be created in which the program module needs to be processed. To do so, right click on **Application** in the project tree and select **Add Object > Task Configuration...**. A dialog opens, which can be used to create a node for several tasks in the project tree. Via **Add** the node is set up in the project tree under **Application > Task Configuration**, which already contains a task. It is automatically opened in the work area.

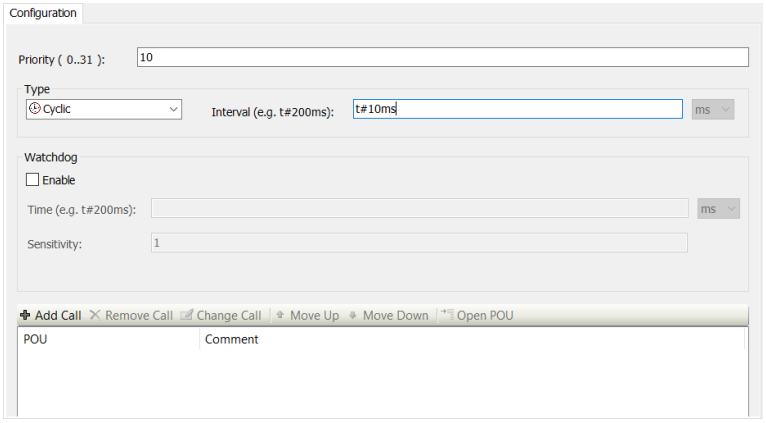


Fig. 20-30: Task configuration

The priority and interval of the task can be set in this window and the desired program module can be appended. The allocation of the POU takes place via **Add + Call**. A dialog opens in which the desired POU is selected and appended to the task.

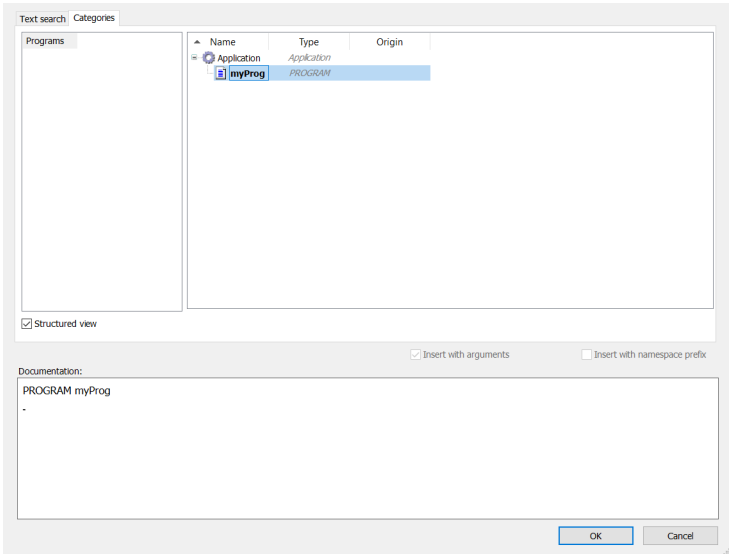


Fig. 20-31: Appending POU to the task

The POU appended to the task is added to the project tree under the node **Application > Task Configuration > Task**.

A watchdog can be configured for the task. The task must be complete before the watchdog expires. The sensitivity of the watchdog can be adjusted.

20.3 Saving u-create studio project

The entire project is saved with the menu command **File > Save Project** or via the Save icon on the toolbar in the directory indicated during the creation of the project. The project is automatically compiled during the saving process.

20.4 Load firmware onto device

There is the possibility to install the firmware on a device (e.g. control) via a removable disk (e.g. SD card). Thereby, the whole control operating system, the runtime system and the application is stored on the removable disk and can be installed on any number of devices.

Therefore the following configuration settings have to be inserted in u-create in the menu **Tools ► Optionen...** under **Software Service**:

Name	Value
Update Server	localhost
System	System version which should be installed on the control or the active static operating panel

To create a removable disk disk it must be plugged into the PC. Then **Create Target** has to be selected in the context menu of the device in the project tree and the following dialog opens:

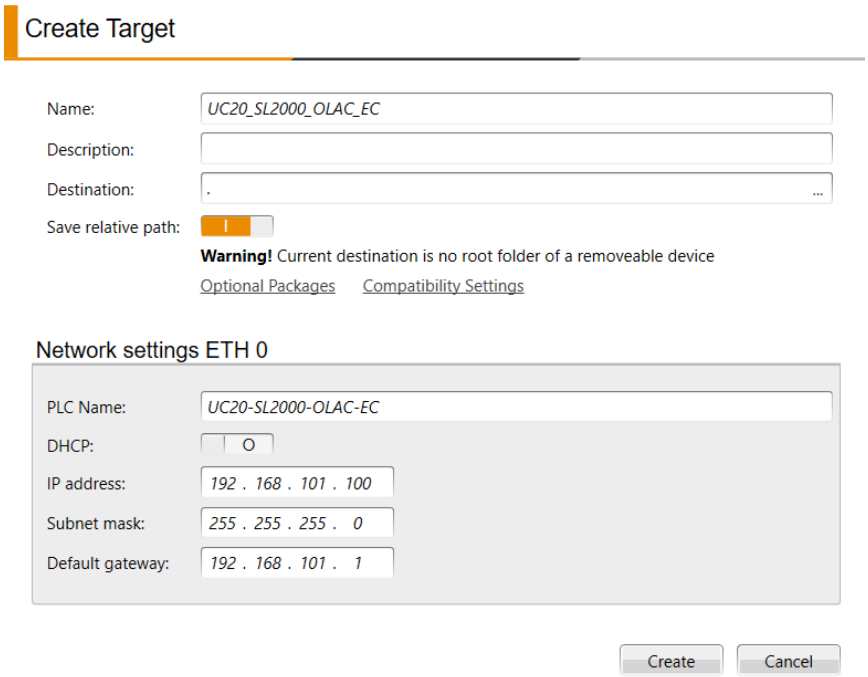


Fig. 20-32: Dialog - Create Target

Name	Description
Name:	Name of the control or the active static operating panel
Description:	Individual description
Destination:	Selection of the removable disk or any folder. If no removable disk is selected, a warning will be shown and the content is saved in the selected directory. If a removable disk is selected, it is checked whether the removable disk is a valid Service Medium. If it is no valid Sevice Medium, a warning is shown and the removable disk can be made valid or bootable via clicking on "Prepare Service Medium (Administrator privileges required)".
Save relative path:	If this option is activated, the path will be stored relative to the project
PLC Name:	Name of the control or the active static operating panel
DHCP:	Use of a DHCP server
IP address:	IP address of the control or the active static operating panel
Subnet mask:	Subnet mask
Default gateway:	Gateway

Optional packages can be used to select additional software components (e.g. OPC-UA). By checking the box for the desired package and OK it will be saved on the removable disk and is available for installation.

A further dialog is opened via **Compatibility settings**, in which a range of serial numbers can be set using "Add". This means that the firmware on the removable disk can only be installed on the devices whose serial numbers lie within this range. This prevents accidental installation on the wrong device.

After all settings are made, the software can be stored on the removable disk via "Create" (dialog "Create target").

Information

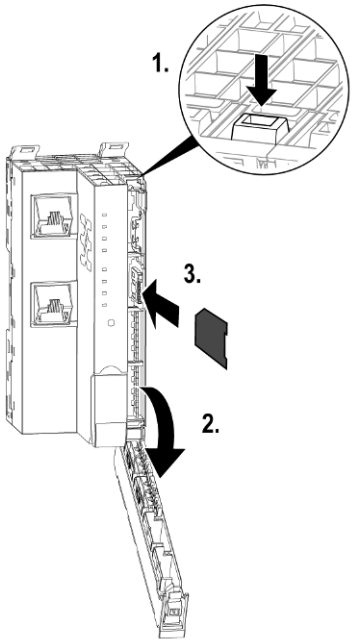
Never remove the removable disk during a storage procedure! This can lead to data loss.

As soon as the display shows "Create target successfully completed ", the removable disk can be removed from the PC and the dialog window can be closed. The software package on the removable disk can now be installed on the desired device.

Installing firmware on devices without display

Caution

Never remove the removable disk during an update procedure nor disconnect the device from the power supply. This can lead to destruction of the firmware on the components and thus make a further operation impossible.



1) Set device curentloess
2) Open front cover
3) Insert SD card

To install the prepared firmware on the removable disk on a device without display, proceed as follows:

1) Set device currentless.
2) Connecting the prepared removalbe disk to the device
3) Perform a restart of the device
4) The update process is carried out and displayed through a flashing orange LED
5) The update process is completely finished as soon as the Run/Stop LED lights green
6) Set device currentless and remove the removable device from the device
7) Perform a restart of the control

The firmware has been installed.

20.5 Configuration of the control

During the next step, the target PLC must be specified and configured. To do so, open the control configuration in u-create studio by double-clicking on the control in the project tree and switch to the "Communication Settings" tab:

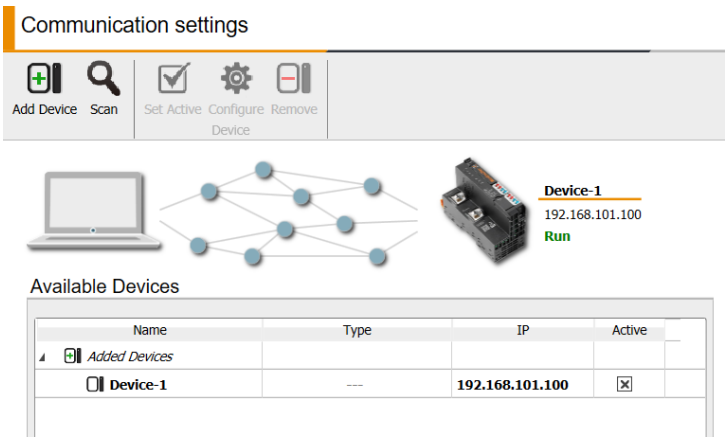


Fig. 20-33: Communication settings

By clicking on **Scan** in the top area of the window, all controls available in the network are displayed.

Select the control intended for your project and click on the button "Set Active". This sets the status of the control in the project tree to "RUN" and the control is highlighted in green.

The download of the translated project described below is applied to this control.

20.6 Compiling project and uploading onto control

The login dialog is opened using the menu command **Online > Selective Download** or by clicking the icon in the toolbar.

Enter the username and password in this dialog (default value: Administrator/tobechanged):

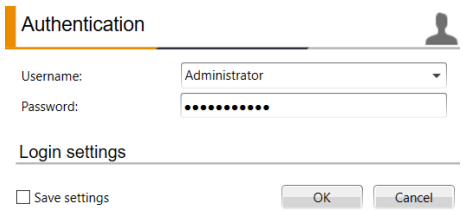


Fig. 20-34: Login dialog

After the login the Selective Download dialog opens.

It can be selected which parts of the project are to be downloaded to the device.

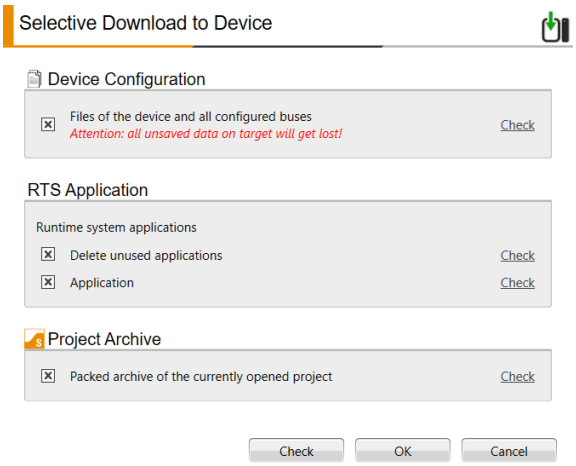


Fig. 20-35: Selection of the data to be uploaded on the control

If a new configuration is loaded onto the control, the control must be restarted. To do this, a dialog is displayed after the configuration is downloaded. If this dialog is confirmed, only the configuration is loaded onto the control and the control implements a restart. The download of the other software components can then be carried out. If the dialog is not confirmed, all required software components are loaded onto the control. The control must then be restarted.

20.7 Starting the project

The downloaded project has not been started yet. To do so, open the module myProg in u-create studio by double-clicking on it and call up the menu command **Debug > Start** (or function key **F5** or ▶ on the toolbar).

The sample program that was just created runs, and the value changes of the variable utilized in the program module are indicated:

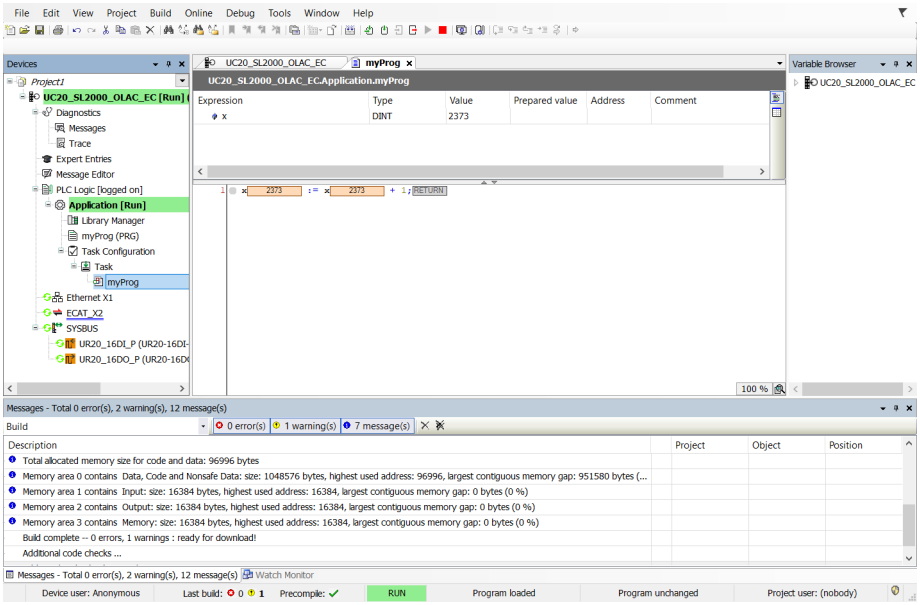


Fig. 20-36: Started sample project

21 Appendix: Addressing in the Ethernet (basics)

Due to the separation of logic and physical protocol levels (Ethernet and TCP/ IP) two types of addresses exist in the network:

- A fixed Ethernet address (MAC-ID) for each unit and
- an IP address, which is allocated to each unit in the network.

The application always sends data to or receives data from an IP address. To ensure their arrival at the receiver a connection must be established between the logic IP address and the physical Ethernet address. The Address Resolution Protocol ARP serves this purpose. An ARP table is stored in each network PC, which specifies the corresponding physical Ethernet address for the IP addresses of the network. If the ARP table does not list an Ethernet address, the IP driver can generally determine it via an ARP request.

Ethernet address (MAC-ID)

MAC-ID (Media Access Control) is the fixed address that clearly identifies an Ethernet device.

IP address

An IP address according to the standard IPv4 is generally specified with 4 decimal numbers divided by points (each 1 byte). Example for an IP address: 192.168.181.1

Both a network and an individual participant are allocated an IP address in the network. The IP address contains:

- The Net ID (specifies a network address) and
- the Host ID (specifies the address of an individual participant in the network). It must be unique, i.e. no two terminal devices can be operated with the same Host ID in the network.

A so-called net mask (subnet mask) is used to determine which numbers of an IP address represent the Net ID and the Host ID.

Net classes

With a "0" as wildcard the net mask for IP addresses defines which bits are used for addressing the participant (Host ID). A "1" as wildcard defines which bits the network address (Net ID) contains. The number of these bits determines which classes the networks belong to:

Net class	Net mask	Description
A	255.0.0.0	Large network
B	255.255.0.0	Medium-sized network
C	255.255.255.0	Small network with a maximum of 254 participants

Example: In a small network with the net mask 255.255.255.0, at the IP address 192.168.181.1 the Net ID is 192.168.181 and the Host ID is 1. If (in a different, medium-sized network) net mask 255.255.0.0 is set, then at IP address 192.169.100.1, the Net ID is 192.169 and the Host ID is 100.1.

Gateway

Networks with different Net IDs are connected together via routers or gateways. If a network participant is to send data to a participant in a different network, the IP address of the gateway must be additionally specified. For addressing the IP three details must be specified:

- IP address
- IP net mask
- IP address of the gateway

Information on the addresses of your in-house network is available from the network administrator.

Setting the IP address

The settings for IP addressing can be manually configured for each terminal unit. In large networks this is done centrally and automatically by means of DHCP (Dynamic Host Configuration Protocol). Here a DHCP server administers the IP addresses and allocates them to the DHCP-capable terminal units. The Weidmüller controls are DHCP-capable. For this, the **Enable DHCP** command must be active in the **Control configuration** in u-create studio. The IP address is then allocated by the DHCP server in the network. During the boot-up the control then requests its IP address from the DHCP server via the network

22 Appendix: Tutorial FoE

FoE (data transfer via EtherCAT) must be done via the PLC application. This chapter shows an example for updating the firmware of the fieldbus coupler UR20-FBC-EC-ECO. The information is displayed in the message window.

```
PROGRAM PRGECatFileTransfer
```

// Declaration

```
VAR
    state: IoApi_State;
    err: UDINT;
    pass: UDINT;
    slavedir: STRING:= '';
    masterdir: STRING:= '/home/admin/FB-EC-FULL-0007674-01_08_00-9.bsc';
    dir: IoEcat_FileDirection:= IoEcat_FileDirection.download;
    mDevHandle: IoApi_Hdl;
    ReqBootStrap: BOOL;
    LeaveBootstrap: BOOL;
    EcatCoupler: K_Io.IO_DEVICE_REF;
    GetState: BOOL;
    SlaveState: IoEcat_State;
    miState: DINT; msState: STRING;
    msFileName: STRING;
    msFileDir: STRING:='/home/admin/";
    mbLoadNewFW: BOOL;
END_VAR
```

// Program blocks

```
(* Requirements:
    Included libraries: K_IoApi / K_SystemCallLibrary / Standard
Description:
    1. Copy the firmware files to "/home/admin" on the
    UC20-SL2000 (e.g. via WinSCP).
    2. Configure device name and firmware file.
    3. Set mbLoadNewFW to TRUE in order to execute firmware update.
*)

//Configure device name and desired firmware file
EcatCoupler := UR20_FBC_EC_ECO;    //Device name at ECAT_X2
msFileName:= 'FB-EC-ECO-0007674-01_00_01-6.bsc';

//Don't change the following code
CASE miState OF

0: //Ready for FW Update
IF mbLoadNewFW THEN
    mDevHandle:= EcatCoupler.DeviceHandle
    mbLoadNewFW:= FALSE;
    miState:= 10;
END_IF

10: //Enter Bootstrap
state:= IoApi_EcatEnterBootstrap(devHdl:= mDevHandle);
IoApi_EcatGetState(devHdl:= mDevHandle,pState:= SlaveState);
IF state = IoApi_State.IoApiStateOk AND
SlaveState = IoEcat_State.Bootstrap THEN
    msState:= CONCAT(EcatCoupler.DeviceName,':Bootstrap entered');
    KSys_TraceUOS(ADR(msState));
    miState:= 11;
END_IF

11: //Download FW-File
dir:= IoEcat_FileDirection.download;
slavedir:= DELETE(msFileDir,msFileName);
masterdir:= CONCAT(msFileDir,msFileName);
```

```
state := K_IoApi.IoApi_EcatFileTransfer(devHdl:= mDevHandle,
direction:= dir,
slaveFileName:= slavedir,
masterFileName:= masterdir,
password:= pass,
errorCode:= err);
IF state = IoApi_State.IoEcatFoEResultSuccess AND err = 0 THEN
    miState:= 20;
    msState:= CONCAT(EcatCoupler.DeviceName,':File downloaded successful');
    KSys_TraceUOS(ADR(msState));
ELSIF err <> 0 THEN
    msState:= CONCAT(EcatCoupler.DeviceName,':Error downloading File');
    KSys_TraceUOS(ADR(msState));
    miState:= 21;
END_IF

20: //Leave Bootstrap
state:= IoApi_EcatLeaveBootstrap(devHdl:= mDevHandle);
IoApi_EcatGetState(devHdl:= mDevHandle,pState:= SlaveState);
IF SlaveState = IoEcat_State.Operational THEN
    msState:= CONCAT(EcatCoupler.DeviceName,':Leaving Bootstrap successful');
    KSys_TraceUOS(ADR(msState));
    miState:= 21;
END_IF

21: //Wait for device
IoApi_EcatGetState(devHdl:= mDevHandle,pState:= SlaveState);
IF SlaveState = IoEcat_State.PreOperational THEN
    miState:= 22;
END_IF

22: //Device OK
IoApi_EcatGetState(devHdl:= mDevHandle,pState:= SlaveState);
IF SlaveState = IoEcat_State.Operational AND NOT EcatCoupler.HasError THEN
    msState:= CONCAT(EcatCoupler.DeviceName,':FW Update successful');
    KSys_TraceUOS(ADR(msState));
    miState:= 0;
END_IF
END_CASE
```

23 Appendix: Tutorial - call C function from IEC

The u-create system offers the possibility to call C functions in IEC created with u-create studio C++.

Information

The function described below is only possible on devices with an "Open-Linux automation control" license.

Therefore a template with a predefinded interface for data exchange exists in u-create studio C++. It is recommanded to create a function in IEC which takes care of the call of the C function and the correct data exchange. The data exchange itself takes place on a shared memory area. Because of this only adresses on the memory area are transfered. The data types from IEC and C are not compatible, so on transfer of the parameters it must be payed attention on the correct type convertion.

Information

There is no automatically type checking. The user himself is responsible for the correct mapping. E.g. if the size of the data types does not match, a wrong memory area is accessed.

The example in this chapter describes the creation of a C function in u-create studio C++ and the call in a IEC application created inu-create studio.

23.1 Preconditions and needed components

The following components are necessary:

- u-create studio C++
- u-create studio
- u-create studio project with an executable IEC application
- Fitting hardware: UC20-SL2000-OLAC-EC, UC20-SL2000-OLAC-EC-CAN

The creation of this example requires basic knowledge in dealing with u-create studio and u-create studio C++.

23.2 Task

In this example application a self made C function should be called via an IEC function in the IEC application. The IEC function provides 3 local variables with different data types (`DWORD`, `STRING[106]` and `REAL`) and a global variable of type `DWORD`. The values of the variables should be changed in the C function as follows:

- The global variable of type `DWORD` should be summated with the variable of type `DWORD`
- The variable of type `REAL` should be multiplicated with 3.
- The variable of type `STRING[106]` should be extended with characters.
- The return value of the C function should be twice as much as the variable of type `DWORD`.

Afterwards, the changed values of the variables should be accessable in the IEC application again.

Therefore the following steps are necessary:

- Creation of the IEC function
- Creation of the C function and download to controller
- Call of IEC function in IEC application and download

23.3 Creating the C function and download

After starting the u-create studio C++ a new project is created via **File ► New ► C/C++ Target Application** in the menu bar.

In the dialog the following parameters are set:

Parameter	Value
Project name	myProject
Location	C:\myProjects
Target	• UC20-SL2000: Control PLC V<VersionNo> ARMHF Linux • KEBA control: Control PLC V<VersionNo> X86 Linux
Project Type	CoDeSys IEC Functions in C

Via **Finish** the project is created and displayed in the project tree.

In this project a C function already exists and the predefined interface is already created.

Under **myIEC_C_Call ► include** in the project tree the file `MyIEC_C_Call.h` is opened with a double click. In this file the structures for the input and output variables are declared respectively and so the interface is adapted to the IEC interface.

In the structure `struct func1_IN` the input parameters of the IEC function are displayed. The number of parameters has to correlate and a correct type mapping has to be done. Therefore the existing code is replaced by the following code in this example:

```
struct func1_IN {
    int32_t *in_dword;
    const char *in_string;
    float *in_real; };
```

In the structure `struct func1_OUT` the output parameters of the IEC function are displayed. The number of parameters has to correlate and a correct type mapping has to be done. Therefore the existing code is replaced with the following code in this example:

```
struct func1_OUT {
    float *out_dword;
    char *out_string; };
```

In the structure `struct func1_Instance` the global variable of the IEC function is displayed. Therefore the existing code is replaced with the following code in this example:

```
struct func1_Instance {
    int32_t *inst_global;
};
```

In addition both files `stdio.h` and `string.h` must be included.

To create the C function the file `MyIEC_C_Call.c` under **myIEC_C_Call ► src** in the project tree must be opened in the working area. The program code in the function `int func1(...)` is removed and replaced with the following code parts.

In the first code line the variable `char msg [106]` is defined. This variable is used later to buffer a text.

In the next codeline the values of the input parameters (addresses of the memory area) which are delivered when calling the C function are displayed in the Trace Monitor of u-create studio .

```
printf("func1 (called from IEC) with params: in_dword:%i, in_string:%s,
in_real:%f \n", *(in->in_dword), in->in_string, *(in->in_real));
```

In the next lines the calculation is executed.

First the variable `inst_global` is added to the variable `in_dword`:

```
*(inst->inst_global) += *(in->in_dword);
```

Then the variable `in_real` is multiplied with 3 and the result is assigned to the variable `out_dword`:

```
*(out->out_dword) = *(in->in_real) * 3;
```

In the next step a text consisting of the variable `in_string` and additional characters is written to the variable `msg`. Then the content of the variable `msg` is copied to the variable `out_string`.

```
sprintf(msg,"C-Function answering to \"%s\" .", in->in_string);
strcpy(out->out_string , msg);
```

In the end the variable `in_dword` is multiplied with 2 and the result is returned as return value of the function.

```
return *(in->in_dword) * 2;
```

The C function has been created.

Now the project is build via **Project ► Build All** in the manu bar and the output (`libMyIEC_C_Call.so`) is loaded to the control in the directory `/appld-isk/application/control/ccontrol/` via **Download**.

23.4 Calling the IEC funtion in the IEC application and download

In the IEC application the created IEC funtion can be used. Therefore a program (e.g. `PLC_PRG`) in u-create studio is extended with the following code:

Local variables

```
VAR
    rv1: DINT;

    res1 : REAL;
    res2 : STRING [106];
END_VAR
```

In the code area the IEC function `MyCFunction` is called as follows:

```
rv1 := MyCFunction( in_dword := 17, in_string := 'Hello c-Function',
in_real := 25.58, out_real => res1, out_string => res2);
```

The variables `res1` and `res2` contain the variables calculated in the C function which can be used later in the application.

Then the project can be loaded to the control (see online help u-create studio) and finally it can be executed.

Index