u-create studio

System manual

Translation of the original instructions
## Record of Revision

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change in chapter</th>
<th>Description</th>
<th>Changed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>06-2019</td>
<td>-</td>
<td>Newly created</td>
<td>hli, hasl</td>
</tr>
<tr>
<td>01</td>
<td>08-2019</td>
<td>Access to flash storage medium</td>
<td>Updated</td>
<td>hasl</td>
</tr>
</tbody>
</table>
# Table of contents

1 **Introduction** ........................................................................................................ 9  
   1.1 Purpose of the document ................................................................................. 9  
   1.2 Requirements ................................................................................................. 9  
   1.3 Intended use ..................................................................................................... 10  
   1.4 Notes on this document ................................................................................... 11  
      1.4.1 Contents of the document .................................................................. 11  
      1.4.2 Not contained in this document ........................................................ 11  
   1.5 Further documentation .................................................................................... 11  
   1.6 Relevant countries and registrations ............................................................. 12  

2 **Safety instructions** ............................................................................................... 13  
   2.1 Representation ............................................................................................... 13  
   2.2 General safety information ............................................................................. 14  
   2.3 Personnel safety .............................................................................................. 14  
   2.4 Safety instructions for programming ............................................................. 14  

3 **System overview** ................................................................................................ 15  
   3.1 Hardware architecture .................................................................................... 16  
   3.2 Software structure ......................................................................................... 16  
   3.3 Network design ............................................................................................... 18  
   3.4 u-control - CPU modules ............................................................................. 19  
   3.5 u-remote peripheral devices ......................................................................... 20  
   3.6 Buses ............................................................................................................... 21  
      3.6.1 EtherCAT ................................................................................................. 21  
      3.6.2 CAN ......................................................................................................... 22  
      3.6.3 Modbus TCP/IP ..................................................................................... 22  
   3.7 u-create studio Toolsuite ............................................................................... 22  
      3.7.1 u-create studio ........................................................................................ 23  
      3.7.2 u-create studio C++ ............................................................................... 24  
      3.7.3 u-create studio Scope ............................................................................. 25  
   3.8 Libraries and interfaces .................................................................................. 25  
      3.8.1 IEC standard libraries ............................................................................. 25  
      3.8.2 u-create studio libraries .......................................................................... 26  
      3.8.3 BasePackage-API (BP API) .................................................................. 26  
      3.8.4 CTRL-API ............................................................................................... 26  
      3.8.5 IO-API ........................................................................................................ 26  
      3.8.6 OPC UA interface ................................................................................. 27
# Table of contents

4 General assembly and installation instructions ........................................ 28

5 Operating behavior .................................................................................. 29
  5.1 Start-up ................................................................................................. 29

6 Software installation .................................................................................. 30
  6.1 Install firmware of the control ............................................................ 30

7 Configuration ............................................................................................ 31
  7.1 Data exchange between control and visualization ............................ 31

8 Program development ................................................................................ 32
  8.1 Task priorities .................................................................................... 32
  8.2 Fast control ....................................................................................... 33
  8.3 Device Service .................................................................................... 34
    8.3.1 Defined state model .................................................................... 34
    8.3.2 Device Service Item .................................................................... 35
    8.3.3 Defined operation behaviour ....................................................... 39
    8.3.4 Status report .............................................................................. 40
    8.3.5 Co-routine model ....................................................................... 40

9 Licensing .................................................................................................. 41
  9.1 Trial license ....................................................................................... 41
    9.1.1 Activation of a trial license (single user license) ....................... 41
    9.1.2 Activation of a trial license (runtime license) ............................ 42
  9.2 Activating a single-user license (software on the PC) ....................... 42
  9.3 Activating a runtime license (software on the device) ...................... 43
  9.4 License status and license overview .................................................. 44
  9.5 Device replacement ............................................................................ 45
    9.5.1 PC replacement (single-user license) ........................................ 45
    9.5.2 Device replacement (runtime license) ......................................... 46
  9.6 Reset the license information ............................................................. 47

10 Device Administration (DevAdmin) ......................................................... 48
  10.1 Tab "Diagnostics" ............................................................................ 48
  10.2 Tab "Configuration" ......................................................................... 50
  10.3 Change password for DevAdmin ...................................................... 52

11 OPC UA Server ....................................................................................... 54
  11.1 Overview of variants ....................................................................... 54
  11.2 Establishing a connection ............................................................... 55
  11.3 Server configuration ....................................................................... 57
  11.4 Generic variable tree ..................................................................... 60
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.5</td>
<td>Creating an information model</td>
<td>61</td>
</tr>
<tr>
<td>11.6</td>
<td>Logging of server operation</td>
<td>68</td>
</tr>
<tr>
<td>12</td>
<td>Software units</td>
<td>70</td>
</tr>
<tr>
<td>12.1</td>
<td>Creating a software unit</td>
<td>70</td>
</tr>
<tr>
<td>12.2</td>
<td>Example meta.xml</td>
<td>71</td>
</tr>
<tr>
<td>13</td>
<td>Data recorder</td>
<td>73</td>
</tr>
<tr>
<td>13.1</td>
<td>Configuration</td>
<td>73</td>
</tr>
<tr>
<td>13.2</td>
<td>Data recording</td>
<td>75</td>
</tr>
<tr>
<td>13.3</td>
<td>Reading out the buffer</td>
<td>79</td>
</tr>
<tr>
<td>13.4</td>
<td>Recording callback from the application</td>
<td>80</td>
</tr>
<tr>
<td>13.5</td>
<td>Data locality and real-time</td>
<td>81</td>
</tr>
<tr>
<td>13.6</td>
<td>Size limitations</td>
<td>81</td>
</tr>
<tr>
<td>13.7</td>
<td>Appendix: C interface</td>
<td>82</td>
</tr>
<tr>
<td>13.7.1</td>
<td>Profile with manual recording</td>
<td>85</td>
</tr>
<tr>
<td>13.7.2</td>
<td>Profile with automatic recording</td>
<td>86</td>
</tr>
<tr>
<td>13.7.3</td>
<td>Profile with triggered recording</td>
<td>87</td>
</tr>
<tr>
<td>13.7.4</td>
<td>Profile with persistence</td>
<td>89</td>
</tr>
<tr>
<td>13.7.5</td>
<td>Evaluation of recordings</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>Diagnostics</td>
<td>94</td>
</tr>
<tr>
<td>14.1</td>
<td>Control diagnosis</td>
<td>94</td>
</tr>
<tr>
<td>14.1.1</td>
<td>Error codes</td>
<td>95</td>
</tr>
<tr>
<td>14.2</td>
<td>Diagnosis data for Weidmüller</td>
<td>95</td>
</tr>
<tr>
<td>14.2.1</td>
<td>Status report</td>
<td>95</td>
</tr>
<tr>
<td>14.2.2</td>
<td>Access to flash storage medium</td>
<td>95</td>
</tr>
<tr>
<td>15</td>
<td>Technical data</td>
<td>99</td>
</tr>
<tr>
<td>16</td>
<td>EC Directives and Standards</td>
<td>100</td>
</tr>
<tr>
<td>17</td>
<td>Appendix: Tutorial - creating an IEC project</td>
<td>101</td>
</tr>
<tr>
<td>17.1</td>
<td>Creating a new project</td>
<td>101</td>
</tr>
<tr>
<td>17.2</td>
<td>Creating a simple program</td>
<td>104</td>
</tr>
<tr>
<td>17.3</td>
<td>Saving u-create studio project</td>
<td>107</td>
</tr>
<tr>
<td>17.4</td>
<td>Load firmware onto device</td>
<td>107</td>
</tr>
<tr>
<td>17.5</td>
<td>Configuration of the control</td>
<td>109</td>
</tr>
<tr>
<td>17.6</td>
<td>Compiling project and uploading onto control</td>
<td>110</td>
</tr>
<tr>
<td>17.7</td>
<td>Starting the project</td>
<td>111</td>
</tr>
<tr>
<td>18</td>
<td>Appendix: Addressing in the Ethernet (basics)</td>
<td>112</td>
</tr>
</tbody>
</table>
Table of contents

19 Appendix: Tutorial - call C function from IEC................................................................. 114
  19.1 Preconditions and needed components................................................................. 114
  19.2 Task .................................................................................................................... 114
  19.3 Creation of IEC function.................................................................................... 115
  19.4 Creating the C function and download............................................................ 116
  19.5 Calling the IEC function in the IEC application and download........................... 118

Index.................................................................................................................................... 119
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 as well 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 combination with a UC20-SL2000-EC 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.

<table>
<thead>
<tr>
<th>Target group</th>
<th>Requisite knowledge and abilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety engineer for &quot;Functional Safety&quot;</td>
<td>Basic technical training (technical college, engineer training or corresponding professional experience).</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>- Principles of functional safety</td>
</tr>
<tr>
<td></td>
<td>- All current standards and safety regulations for the machine/plant, in particular also knowledge of validation conforming to EN ISO 13849-2.</td>
</tr>
<tr>
<td></td>
<td>- Special risk potential of the machine/system and the production process</td>
</tr>
<tr>
<td></td>
<td>- Specific protective measures to avert machine-specific hazards (based on hazard and risk analysis)</td>
</tr>
<tr>
<td></td>
<td>- Functioning and application limits of the safety components (including safety PLC).</td>
</tr>
<tr>
<td></td>
<td>- In-depth knowledge of national accident prevention regulations.</td>
</tr>
<tr>
<td>Project engineer</td>
<td>Technical basic education (advanced technical education, engineering degree or corresponding professional experience).</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>- The method of operation of a PLC,</td>
</tr>
<tr>
<td></td>
<td>- Current valid safety information,</td>
</tr>
<tr>
<td></td>
<td>- The application.</td>
</tr>
<tr>
<td>Target group</td>
<td>Requisite knowledge and abilities</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Electrician</td>
<td>Technical training in electrical engineering (based on industry-standard training guidelines).</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>● Current valid safety information,</td>
</tr>
<tr>
<td></td>
<td>● Wiring guidelines</td>
</tr>
<tr>
<td></td>
<td>● Circuit diagrams</td>
</tr>
<tr>
<td></td>
<td>● System analysis and troubleshooting</td>
</tr>
<tr>
<td></td>
<td>● Professional manufacture of electrical connections in conformity with national and international regulations.</td>
</tr>
<tr>
<td>Programmer</td>
<td>Technical training (University of Applied Science, engineering degree or corresponding professional experience).</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>● The method of operation of a PLC,</td>
</tr>
<tr>
<td></td>
<td>● Current valid safety information,</td>
</tr>
<tr>
<td></td>
<td>● Programming a PLC (IEC61131).</td>
</tr>
<tr>
<td>Start-up operator</td>
<td>Technical basic education (advanced technical education, engineering degree or corresponding professional experience),</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>● Current valid safety information,</td>
</tr>
<tr>
<td></td>
<td>● Method of operation of the machine or system</td>
</tr>
<tr>
<td></td>
<td>● Fundamental functions of the application</td>
</tr>
<tr>
<td></td>
<td>● System analysis and troubleshooting</td>
</tr>
<tr>
<td></td>
<td>● Setting options on the operating devices</td>
</tr>
<tr>
<td>Service technician</td>
<td>Technical basic education (advanced technical education, engineering degree or corresponding professional experience),</td>
</tr>
<tr>
<td></td>
<td>Knowledge of:</td>
</tr>
<tr>
<td></td>
<td>● The method of operation of a PLC,</td>
</tr>
<tr>
<td></td>
<td>● Current valid safety information.</td>
</tr>
<tr>
<td></td>
<td>● Method of operation of the machine or system</td>
</tr>
<tr>
<td></td>
<td>● Diagnostic options</td>
</tr>
<tr>
<td></td>
<td>● Systematic error analysis and troubleshooting</td>
</tr>
</tbody>
</table>

### 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:
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 operating elements and displays
- Description of the assembly and installation
- Operating behavior
- Description of the software installation
- Description of the software interfaces
- Diagnosis and maintenance
- Technical data
- EMC and wiring guidelines
- Terminology

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. When using safe I/O modules or safe power supply modules, please also refer to the Modules for Functional Safety 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.

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 programming

**CAUTION!**

- The instructions contained in these additional manuals provided must be precisely followed in all circumstances. If not observed, sources of danger may arise and suppressors integrated into the u-create may be rendered ineffective.
- Aside from the safety instructions given in these manuals, the 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, no dangerous operating conditions must 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 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 IO 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**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>... Control</td>
</tr>
<tr>
<td>2</td>
<td>... UR20 module</td>
</tr>
<tr>
<td>3</td>
<td>... SD card</td>
</tr>
<tr>
<td>4</td>
<td>... PWM module</td>
</tr>
<tr>
<td>5</td>
<td>... DC motor</td>
</tr>
<tr>
<td>6</td>
<td>... UR67 module</td>
</tr>
<tr>
<td>7</td>
<td>... UR20 station</td>
</tr>
<tr>
<td>8</td>
<td>... Tool-Suite</td>
</tr>
<tr>
<td>9</td>
<td>... Touch panel</td>
</tr>
<tr>
<td>10</td>
<td>... Ethernet</td>
</tr>
<tr>
<td>11</td>
<td>... Field bus</td>
</tr>
</tbody>
</table>
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.

All 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 IO 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.
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 and packages generated by Weidmüller 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.

**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
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 1740 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.

IP addresses of the control system components are completely independent from the IP addresses of the network. The following values are factory preset by default.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control (X1):</td>
<td>192.168.101.100</td>
</tr>
<tr>
<td>Subnet mask:</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>Gateway:</td>
<td>192.168.101.x</td>
</tr>
<tr>
<td>u-view Panel:</td>
<td>192.168.101.x</td>
</tr>
</tbody>
</table>

**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.

![Diagram of service PC in the machine network]

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

### 3.4 u-control - CPU modules

<table>
<thead>
<tr>
<th>Designation</th>
<th>Performance data</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC20-SL2000-EC</td>
<td>Dual Core A9, 512 MB RAM, 8 GB EMMC</td>
<td>• u-remote bus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• EtherCAT Master</td>
</tr>
</tbody>
</table>

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

u-control can be connected directly to the following u-remote I/O modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Order No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UR20-4DI-P</td>
<td>1315170000</td>
</tr>
<tr>
<td>UR20-8DI-P</td>
<td>1315180000</td>
</tr>
<tr>
<td>UR20-8DI-P-3W-HD</td>
<td>1315190000</td>
</tr>
<tr>
<td>UR20-16DI-P</td>
<td>1315200000</td>
</tr>
<tr>
<td>UR20-16DI-P-PLC-INT</td>
<td>1315210000</td>
</tr>
<tr>
<td>UR20-4DO-P</td>
<td>1315220000</td>
</tr>
<tr>
<td>UR20-4DO-P-2A</td>
<td>1315230000</td>
</tr>
<tr>
<td>UR20-8DO-P</td>
<td>1315240000</td>
</tr>
<tr>
<td>UR20-16DO-P</td>
<td>1315250000</td>
</tr>
<tr>
<td>UR20-16DO-P-PLC-INT</td>
<td>1315270000</td>
</tr>
<tr>
<td>UR20-4DI-N</td>
<td>1315350000</td>
</tr>
<tr>
<td>UR20-8DI-N-3W</td>
<td>1315370000</td>
</tr>
<tr>
<td>UR20-16DI-N</td>
<td>1315390000</td>
</tr>
<tr>
<td>UR20-16DI-N-PLC-INT</td>
<td>1315400000</td>
</tr>
<tr>
<td>UR20-4DO-N</td>
<td>1315410000</td>
</tr>
<tr>
<td>UR20-4DO-N-2A</td>
<td>1315420000</td>
</tr>
<tr>
<td>UR20-8DO-N</td>
<td>1315430000</td>
</tr>
<tr>
<td>UR20-16DO-N</td>
<td>1315440000</td>
</tr>
<tr>
<td>UR20-16DO-N-PLC-INT</td>
<td>1315450000</td>
</tr>
<tr>
<td>UR20-4RO-SSR-255</td>
<td>1315540000</td>
</tr>
<tr>
<td>UR20-4RO-CO-255</td>
<td>1315550000</td>
</tr>
<tr>
<td>UR20-1CNT-100-1DO</td>
<td>1315570000</td>
</tr>
<tr>
<td>UR20-1CNT-500</td>
<td>1315580000</td>
</tr>
<tr>
<td>UR20-2CNT-100</td>
<td>1315590000</td>
</tr>
<tr>
<td>UR20-2PWM-PN-0.5A</td>
<td>1315600000</td>
</tr>
<tr>
<td>UR20-2PWM-PN-2A</td>
<td>1315610000</td>
</tr>
<tr>
<td>UR20-4AI-UI-16</td>
<td>1315620000</td>
</tr>
<tr>
<td>UR20-8AI-I-16-HD</td>
<td>1315650000</td>
</tr>
<tr>
<td>UR20-8AI-I-PLC-INT</td>
<td>1315670000</td>
</tr>
<tr>
<td>UR20-4AO-UI-16</td>
<td>1315680000</td>
</tr>
<tr>
<td>UR20-4AI-RTD-DIAG</td>
<td>1315700000</td>
</tr>
<tr>
<td>UR20-4AI-TC-DIAG</td>
<td>1315710000</td>
</tr>
<tr>
<td>UR20-1COM-232-485-422</td>
<td>1315750000</td>
</tr>
<tr>
<td>UR20-PF-O-1DI-SIL</td>
<td>1335030000</td>
</tr>
<tr>
<td>UR20-PF-O-2DI-DELAY-SIL</td>
<td>1335040000</td>
</tr>
<tr>
<td>UR20-PF-O-2DI-SIL</td>
<td>1335050000</td>
</tr>
<tr>
<td>Module</td>
<td>Order No.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>UR20-4AI-UI-12</td>
<td>1394390000</td>
</tr>
<tr>
<td>UR20-8DI-P-3W</td>
<td>1394400000</td>
</tr>
<tr>
<td>UR20-4DO-PN-2A</td>
<td>1394420000</td>
</tr>
<tr>
<td>UR20-2DI-P-TS</td>
<td>1460140000</td>
</tr>
<tr>
<td>UR20-4DI-P-TS</td>
<td>1460150000</td>
</tr>
<tr>
<td>UR20-4AI-UI-16-HD</td>
<td>1506920000</td>
</tr>
<tr>
<td>UR20-2FCNT-100</td>
<td>1508080000</td>
</tr>
<tr>
<td>UR20-1SSI</td>
<td>1508090000</td>
</tr>
<tr>
<td>UR20-8DO-P-2W-HD</td>
<td>1509830000</td>
</tr>
<tr>
<td>UR20-4AO-UI-16-HD</td>
<td>1510690000</td>
</tr>
<tr>
<td>UR20-4DI-2W-230V-AC</td>
<td>1550070000</td>
</tr>
<tr>
<td>UR20-4AI-R-HS-16-DIAG</td>
<td>2001670000</td>
</tr>
<tr>
<td>UR20-3EM-230V-AC</td>
<td>2007420000</td>
</tr>
<tr>
<td>UR20-4DI-P-3W</td>
<td>2009360000</td>
</tr>
<tr>
<td>UR20-4AO-UI-16-M</td>
<td>2453880000</td>
</tr>
<tr>
<td>UR20-4AI-RTD-HP-DIAG</td>
<td>2456540000</td>
</tr>
<tr>
<td>UR20-8DI-ISO-2W</td>
<td>2457240000</td>
</tr>
<tr>
<td>UR20-4DO-ISO-4A</td>
<td>2457250000</td>
</tr>
<tr>
<td>UR20-8AI-RTD-DIAG-2W</td>
<td>2555940000</td>
</tr>
</tbody>
</table>

The modules not directly supported on the system bus can alternatively be used via UR20-FBC-EC.

For further information please refer to the u-remote manual or the product catalogue (Automation section).

3.6 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
- CAN

The following slave fieldbus interfaces are available depending on the control version:
- Modbus TCP/IP

3.6.1 EtherCAT

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

The availability of CAN interfaces in the system is realized by adding a FM 200/A on this CPU module or on a bus link module.

When using the CAN interface, make sure to use u-create studio with CAN controller.

3.6.3 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.

3.7 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++:**
  The 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:**
  The 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.

- **u-create studio UosDiagnostic: for expert diagnosis:**
UosDiagnostic is a tool for diagnosis and is used to display expert information.
The following chapters describe the tools in more detail.

3.7.1 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. There are two textual and three graphic programming languages. 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)".

<table>
<thead>
<tr>
<th>Programming language</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured text (ST)</td>
<td>Textual programming languages</td>
<td>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 (&quot;IF..THEN..ELSE&quot;) or in loops (WHILE..DO).</td>
</tr>
<tr>
<td>Function block diagram (FBD)</td>
<td></td>
<td>Allow programming with logical symbols. It is very suitable for logic controls.</td>
</tr>
<tr>
<td>Continuous graphic Function Chart (CFC)</td>
<td>Graphic programming languages</td>
<td>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.</td>
</tr>
<tr>
<td>Ladder Diagram (KOP)</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

3.7.2 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 IO 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.

### 3.7.3 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 IEC programs 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.

### 3.8 Libraries and interfaces

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

#### 3.8.1 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
3.8.2 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.

3.8.3 BasePackage-API (BP API)

The BasePackage interface is implemented as an IEC and C interface. It is used as the control access for standalone non-realtime programs and expansions for C programs with limited realtime capabilities.

Usage options:

- Access to system variables
- Access to message system
- Triggering of the status report
- Longterm data recording

A detailed description of the interface can be found in the online help for u-create studio.

3.8.4 CTRL-API

This interface is used to enable C variables of a C program to be able to use them on the system. In addition, the variables can be made available to the software oscilloscope (SWO).

Usage options:

- Access to catalog entries
- Creation of log files

A detailed description of the interface can be found in the online help for u-create studio.

3.8.5 IO-API

This interface is used for high-performance access of a C program to IO endpoints on modules or drive units.

Usage options:

- Access to system variables
- Access to EtherCAT and CAN
A detailed description of the interface can be found in the online help for u-create studio.

3.8.6 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 detailed 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 this 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>).
4 General assembly and installation instructions

Caution
Improper handling can damage the modules, option modules and the control system.

- Switch off the operating and on-load power supplies before carrying out assembly, installation or maintenance work.

Caution
Damage to components!
Handle all modules and components with care. Please ensure the following:

- Clean contact surfaces (to avoid contact faults).
- Bus plugs that are not bent.
- Ensure that no pieces of wires, fillings or swarf fall into the device when you are drilling holes or connecting wires.

Detailed information on assembly and installation instructions, as well as dimensions and minimum distances, can be found in the project engineering handbook for the particular module.
5 Operating behavior

5.1 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.
- Correctly formatted storage medium with the system software on it is in the slot of the control.
6 Software installation

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

To install the required u-create components on your PC, proceed as follows:
1) Open the provided installation directory.
2) Open the "Setup" subdirectory.
3) Run the application "setup.exe".
The installation dialog is opened.
4) Follow the instructions in the installation dialog.
Once all components have been installed, there is a link to a directory on the desktop, from where all the necessary components can be started.

Information
All running applications must be completed during the installation process.

6.1 Install firmware of the control

No firmware besides the hardware related software is installed on the control at delivery status. This firmware (without hardware related software) must be loaded to a storage medium via u-create studio and with this storage medium it must be installed on the control.

Therefore an empty storage medium with at least 2 GB memory which is plugged into the PC is needed.

For more information see online help of u-create studio chapter "Create target disk".
7 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 EA 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 via Service App "DevAdmin" (see 10 Device Administration (DevAdmin)).

7.1 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).
8 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.

8.1 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

<table>
<thead>
<tr>
<th>Priority linux</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>99 - 82</td>
<td>Tasks of this priority are reserved for system and for robotic system add-on</td>
</tr>
</tbody>
</table>

High prior real time tasks (default priority 10)

<table>
<thead>
<tr>
<th>Priority linux</th>
<th>Priority IEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>2</td>
</tr>
<tr>
<td>72</td>
<td>3</td>
</tr>
<tr>
<td>69</td>
<td>4</td>
</tr>
<tr>
<td>66</td>
<td>5</td>
</tr>
<tr>
<td>63</td>
<td>6</td>
</tr>
</tbody>
</table>

Low prior real time tasks

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

<table>
<thead>
<tr>
<th>Priority linux</th>
<th>Priority IEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>7</td>
</tr>
<tr>
<td>Priority linux</td>
<td>Priority IEC</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>57</td>
<td>8</td>
</tr>
<tr>
<td>54</td>
<td>9</td>
</tr>
<tr>
<td>51</td>
<td>10</td>
</tr>
<tr>
<td>48</td>
<td>11</td>
</tr>
<tr>
<td>46</td>
<td>Reserved for UOS task</td>
</tr>
<tr>
<td>45</td>
<td>12</td>
</tr>
<tr>
<td>42</td>
<td>13</td>
</tr>
<tr>
<td>39</td>
<td>14</td>
</tr>
<tr>
<td>36</td>
<td>15</td>
</tr>
</tbody>
</table>

**Tasks outside real time**

This tasks run in the linux scheduler "SCHED_OTHER".

<table>
<thead>
<tr>
<th>Priority linux</th>
<th>Priority IEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (&quot;nice level&quot;)</td>
<td>16 - 31</td>
</tr>
<tr>
<td>0 (&quot;nice level&quot;)</td>
<td>Default linux processes</td>
</tr>
</tbody>
</table>

### 8.2 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.

The configuration of the fast control happens in the u-create studio (see on-line help "Configure EtherCAT slave").
8.3 **Device Service**

The DeviceService is a software service that 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`.

### 8.3.1 Defined state model

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

![DeviceService state model](image)

**States**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OsReady</td>
<td>The operating system is ready and the Device Service is available. Programs not initialized.</td>
</tr>
<tr>
<td>Stop</td>
<td>The programs are running, Applications not running</td>
</tr>
<tr>
<td>Run</td>
<td>Programs and applications are running</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Command type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Controller</td>
<td>State transition</td>
<td>This status transition is run automatically once the device is started. The status changes from &quot;OsReady&quot; to &quot;Run&quot;.</td>
</tr>
</tbody>
</table>
### 8.3.2 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.

#### 8.3.2.1 Defined state model Device Service Item

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

![DeviceService Item status model](image)

Fig. 8-6: 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:

<table>
<thead>
<tr>
<th>Status transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init DeviceService</td>
<td>The DeviceService is started and imports all programs created (DeviceService Items)</td>
</tr>
<tr>
<td>Init Controller</td>
<td>The program must execute all the necessary initializations and then change to the Stop process</td>
</tr>
<tr>
<td>Start Applications</td>
<td>The program begins processing and must change to the Run status</td>
</tr>
<tr>
<td>Stop Applications</td>
<td>The program ends processing and must change to the Stop status</td>
</tr>
<tr>
<td>Exit Controller</td>
<td>The program must end all initializations carried out in &quot;Init Controller&quot; and change to the OS-Ready status.</td>
</tr>
<tr>
<td>ExitDeviceService</td>
<td>The DeviceService ends all programs and itself</td>
</tr>
</tbody>
</table>

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

#### 8.3.2.2 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the program (DeviceServiceItem)</td>
</tr>
<tr>
<td>variables</td>
<td>Global variables of the program</td>
</tr>
<tr>
<td>defaultvariables</td>
<td>Global variables with default values that can be overwritten</td>
</tr>
</tbody>
</table>

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

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

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

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:

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

Example file `example.item.ex`:

```lua
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
8.3.3 Defined operation behaviour

Each DeviceServiceItem (program) can provide the following functions:

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

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.
8.3.4 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.

8.3.5 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.
9 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.

9.1 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.

9.1.1 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.
9.1.2 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 License ► Trial license.

![License tab](image)

4) Click on Activate trial license.

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

9.2 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) If necessary, select the license for the desired product.
5) Click on **Activate Selected Licenses now**, and the licensing process is executed.

6) Confirm the dialog with **OK**.
   The license has been successfully activated.

### 9.3 Activating a runtime license (software on the device)

The following options are available for activating a license on the device:
- Online activation using "DevAdmin"
Online activation using "DevAdmin"

For this online activation, a PC with an Internet connection is required. The device on which the software to be licensed is installed does not require an Internet connection, but it must be connected to the PC via a network connection.

Proceed as follows to activate the license:

1) Establish a connection between the PC and the device using DevAdmin.
2) Click on License ▶ Online Activation.
3) Enter the ticket code.
4) Click on Activate.

The license was successfully activated on the device.

9.4 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 License ▶ Overview tab, an overview of all licenses and their validity periods as well as the license type is shown.
9.5 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 times. Afterwards, Weidmüller must be contacted.

9.5.1 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:

- License rehost: The license can be returned by the existing PC, which is still functional.
- Reactivation of 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.

Rehosting the license

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

<table>
<thead>
<tr>
<th>My Licenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>ReControl Simulation</td>
</tr>
</tbody>
</table>

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".

### Reactivating 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 **Restore Licenses**.
4) Accept the restore conditions.
5) Click on **Restore Selected Licenses Now**.

The reactivated license was successfully activated on the PC.

### 9.5.2 Device replacement (runtime license)

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

- Online restore of a license using DevAdmin
- Offline restore of a license using DevAdmin

The original license runs on the new device for an additional 14 days. The license can only be used 2 x on a new device.

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

### 9.6 Reset the license information

In certain cases it is necessary 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 License ► 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.
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 an internet browser and inserting the address http://<IP address of the control>.

For a safe and encrypted transmission of the login data in the internal machine network a HTTPS connection is used. Since a SSL certificate of a certification body is tied either on a fixed IP address or on a domain name the device contains a self-signed certificate. This causes a safety warning on current web browsers.

The following shows examples for accepting the certificate in common web browsers. The procedure can differ in future browser versions.

- Microsoft Internet Explorer: After input of the IP address click "Continue to this website (not recommended)".
- Mozilla Firefox: Select „Advanced” and then "Add Exception...". In the subsequent window "Confirm Security Exception" is selected. Due to the option "Permanently store this exception" the exception is stored for this certain IP address.
- Google Chrome: First click "ADVANCED" and then "Further to <IP> (insecure)". The exception is saved automatically.

The user is logged in with the following login data:
Username: Administrator
Password: tobechanged

The DevAdmin has the following tabs:

- **Diagnostics**: Read information of the control, trigger state report or crash report
- **Configuration**: Do network settings, time settings and time area settings
- **Backup/Restore**: Create or import a backup

10.1 Tab "Diagnostics"

The tab "Diagnostics" is divided into the following areas:

- Device Information
- Device State
- Statereport
- Crashreport
Area "Device Information"

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

<table>
<thead>
<tr>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Name</td>
<td>Designation of the control</td>
</tr>
<tr>
<td>Ser.No.</td>
<td>Serial number of the control</td>
</tr>
<tr>
<td>Part No.</td>
<td>Material number of the control</td>
</tr>
<tr>
<td>Rev.</td>
<td>Revision number of the control</td>
</tr>
<tr>
<td>Operating Hours</td>
<td>Operating hours of the control</td>
</tr>
<tr>
<td>Workload</td>
<td>Current utilization of the control</td>
</tr>
<tr>
<td>Temperature</td>
<td>Current temperature of the control</td>
</tr>
<tr>
<td>Battery</td>
<td>State of the battery</td>
</tr>
</tbody>
</table>
Area "Device State"

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

<table>
<thead>
<tr>
<th>Information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device State</td>
<td>Status of the control</td>
</tr>
</tbody>
</table>

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 Download statereport from device a state report on the control can be selected and downloaded to the PC.

Area "Crashreport"

Via No crashreport available a crash report on the control can be selected and downloaded to the PC.

10.2 Tab "Configuration"

The tab "Configuration" is divided into the following areas:

- Network Hostname
- Interface Ethernet0
- Time, Date and Timezone
Fig. 10-8: DevAdmin - Configuration

**Area "Network Hostname"**

In the area "Hostname" the following settings can be done:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>Host name of the control</td>
</tr>
</tbody>
</table>

**Area "Interface Ethernet0"**

In the area "Interface Ethernet0" the following settings can be done for the physical interface X1:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DHCP</td>
<td>Activate or deactivate DHCP</td>
</tr>
<tr>
<td>IP Address</td>
<td>IP address of the control</td>
</tr>
<tr>
<td>Netmask</td>
<td>Set subnet mask</td>
</tr>
</tbody>
</table>
### Setting Description

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway</td>
<td>Set gateway</td>
</tr>
<tr>
<td>DNS-Server 1</td>
<td>Set DNS server 1</td>
</tr>
<tr>
<td>DNS-Server 2</td>
<td>Set DNS server 2</td>
</tr>
</tbody>
</table>

### Area "Time, Date and Timezone"

In the area "Time, Date and Timezone" the following settings can be done:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Set clock</td>
</tr>
<tr>
<td>Date</td>
<td>Set date</td>
</tr>
<tr>
<td>Area</td>
<td>Select region</td>
</tr>
<tr>
<td>City/Zone</td>
<td>Select city</td>
</tr>
</tbody>
</table>

### Area "Long-term Diagnostic Monitor"

In the area "Long-term Diagnostic Monitor" you can activate or deactivate the long-term recording of device data.

With **Apply all changes** all settings are applied and with **Discard all changes** they are discarded.

### 10.3 Change password for DevAdmin

In the top right area of the **DevAdmin** the button **Change Password** is located.

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

![Change password window](image-url)

**Fig. 10-9: DevAdmin - Change Password**

The following data are inserted into the fields:
Current password: 
New password: 
Reenter new password: 

Via **Apply** the new password is applied.
Via **Cancel** the procedure is gets aborted.
11 OPC UA Server

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.

![Overview](Fig. 11-10: 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.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>OPC UA Server</th>
<th>OPC UA Server Pro</th>
<th>OPC UA Server Adv</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecureChannel:</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Number of simultaneous client connections (sessions):</td>
<td>1</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>User authentication:</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Services as per table below:</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Generic variable tree:</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Functionality | OPC UA Server | OPC UA Server Pro | OPC UA Server Adv
--- | --- | --- | ---
Creating an information model: | 1 | | several\(^1\)
Support of customer-specific events and methods: | 1\(^1\) | | several\(^1\)

\(^1\) These functionalities are not currently supported yet.

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

<table>
<thead>
<tr>
<th>Service Sets as per OPC UA Specification Part 4</th>
<th>Supported services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery</td>
<td>complete</td>
</tr>
<tr>
<td>SecureChannel</td>
<td>complete</td>
</tr>
<tr>
<td>Session</td>
<td>complete</td>
</tr>
<tr>
<td>Nodemanagement</td>
<td>not supported</td>
</tr>
</tbody>
</table>
| 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`:

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

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:

<table>
<thead>
<tr>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Server port</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>Application name</td>
</tr>
<tr>
<td>ApplicationURI</td>
<td>Application URI</td>
</tr>
<tr>
<td>ProductURI</td>
<td>Product URI</td>
</tr>
<tr>
<td>ProductName</td>
<td>Product name</td>
</tr>
<tr>
<td>ManufacturerName</td>
<td>Manufacturer name</td>
</tr>
</tbody>
</table>
**Designation** | **Description**
---|---
ShowVariableTree | Display of generic variable tree (1 = activated, 0 = deactivated)

**Example**

```
[OpcUa]
[OpcUa.Application]
Port=4840
ApplicationName="SampleApplicationName"
ApplicationURI="SampleApplicationURI"
ProductURI="http://www.company.com/"
ProductName="OPC-UA Server"
ManufacturerName="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.

<table>
<thead>
<tr>
<th>Security Policy</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic256Sha256</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Mode</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>1</td>
</tr>
<tr>
<td>Sign and Encrypt</td>
<td>2</td>
</tr>
</tbody>
</table>

**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
Policy=3
MessageMode=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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinPublishingInterval</td>
<td>Minimum time interval in milliseconds, data type Double</td>
</tr>
<tr>
<td>MaxPublishingInterval</td>
<td>Maximum time interval in milliseconds, data type Double</td>
</tr>
</tbody>
</table>

**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**

```plaintext
[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:

```plaintext
[OpcUa.Subscription.Publishing]
Interval=100 // int32 - ms
Interval=20 // int32 - ms
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns</td>
<td>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 &quot;NamespaceArray&quot; object.</td>
</tr>
<tr>
<td>isDefaultNs</td>
<td>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.</td>
</tr>
</tbody>
</table>

In the following example, two namespaces are created:

```xml
<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.

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

```xml
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DataType</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>NodeId</code></td>
<td>Defines the node ID of the current instance, and this must be unique. This server version only supports numeric node IDs.</td>
</tr>
<tr>
<td><code>BrowseName</code></td>
<td>Internally used name for node addresses.</td>
</tr>
<tr>
<td><code>DisplayName</code></td>
<td>Displayed name for the OPC UA client.</td>
</tr>
<tr>
<td><code>AccessLevel</code></td>
<td>Defines the access rights for the <code>Value</code> attribute of a variable. By default, no access is allowed. The attribute is specified as hexadecimal number and interpreted as <code>AccessLevelType</code> defined in the OPC UA specification 3. Example: &quot;0x01&quot; = CurrentRead, &quot;0x03&quot; = CurrentRead and CurrentWrite.</td>
</tr>
<tr>
<td><code>ValueConstant</code></td>
<td>Allocation of a constant.</td>
</tr>
<tr>
<td><code>ValuePath</code></td>
<td>Path to a system variable. The path can be displayed in the u-create studio browser after successful login.</td>
</tr>
</tbody>
</table>

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 `NamespaceUris` is defined in the entry).

```xml
<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:

```xml
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueRank=&quot;1&quot;</td>
<td>Determines the number of array dimensions. Only the value &quot;1&quot; is supported.</td>
</tr>
</tbody>
</table>
In the following example, the entire IEC array is instantiated in the server as an OPC UA array.

```xml
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayOffset</td>
<td>Specifies the start value of the array subrange (left index).</td>
</tr>
<tr>
<td>ArrayLength</td>
<td>Desired length of the array subrange.</td>
</tr>
</tbody>
</table>

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.

```xml
<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

```xml
<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"/>
    </Object>
  </Instances>
</OpcUAInformationModel>
```
<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:SByte</DataType>
   <NodeId>i=11008</NodeId>
   <BrowseName>MyInt8</BrowseName>
   <DisplayName>MyInt8</DisplayName>
   <ValuePath>APPL.system.vSINT</ValuePath>
   <AccessLevel>0x01</AccessLevel>
</Variable>

<!-- variable is only readable -->

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int16</DataType>
   <NodeId>i=11009</NodeId>
   <BrowseName>MyInt16</BrowseName>
   <DisplayName>MyInt16</DisplayName>
   <ValuePath>APPL.system.vINT</ValuePath>
   <AccessLevel>0x01</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int32</DataType>
   <NodeId>i=11010</NodeId>
   <BrowseName>MyInt32</BrowseName>
   <DisplayName>MyInt32</DisplayName>
   <ValuePath>APPL.system.vDINT</ValuePath>
   <AccessLevel>0x01</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int64</DataType>
   <NodeId>i=11011</NodeId>
   <BrowseName>MyInt64</BrowseName>
   <DisplayName>MyInt64</DisplayName>
   <ValuePath>APPL.system.vLINT</ValuePath>
   <AccessLevel>0x01</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Float</DataType>
   <NodeId>i=11013</NodeId>
   <BrowseName>MyFloat</BrowseName>
   <DisplayName>MyFloat</DisplayName>
   <ValuePath>APPL.system.vREAL</ValuePath>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Double</DataType>
   <NodeId>i=11014</NodeId>
   <BrowseName>MyDouble</BrowseName>
   <DisplayName>MyDouble</DisplayName>
   <ValuePath>APPL.system.vLREAL</ValuePath>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Boolean</DataType>
   <NodeId>i=11016</NodeId>
   <BrowseName>MyBool</BrowseName>
   <DisplayName>MyBool</DisplayName>
   <ValuePath>APPL.system.vBOOL</ValuePath>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Boolean</DataType>
   <NodeId>i=11017</NodeId>
   <BrowseName>MyDateTime</BrowseName>
   <DisplayName>MyDateTime</DisplayName>
   <ValuePath>APPL.system.vDATE_AND_TIME</ValuePath>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:String</DataType>
   <NodeId>i=11018</NodeId>
   <BrowseName>MyString</BrowseName>
   <DisplayName>MyString</DisplayName>
   <ValuePath>APPL.system.vSTRING</ValuePath>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<!-- string constant -->

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:String</DataType>
   <NodeId>i=11020</NodeId>
   <BrowseName>String Constant</BrowseName>
   <DisplayName>String Constant</DisplayName>
   <ValueConstant>My test string constant</ValueConstant>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<!-- int32 constant -->

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Integer.0:Int32</DataType>
   <NodeId>i=11021</NodeId>
   <BrowseName>Int32 Constant</BrowseName>
   <DisplayName>Int32 Constant</DisplayName>
   <ValueConstant>12</ValueConstant>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<!-- float constant -->

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Number.0:Float</DataType>
   <NodeId>i=11022</NodeId>
   <BrowseName>Float Constant</BrowseName>
   <DisplayName>Float Constant</DisplayName>
   <ValueConstant>12.47</ValueConstant>
   <AccessLevel>0x03</AccessLevel>
</Variable>

<!-- create a subfolder for arrays -->

<Object>
   <DataType>0:Types.0:ObjectTypes.0:BaseObjectType.0:FolderType</DataType>
   <NodeId>i=11023</NodeId>
   <BrowseName>1-dim Arrays</BrowseName>
   <DisplayName>1-dim Arrays</DisplayName>
</Object>

<!-- boolean array - block mapping -->

<Variable>
   <DataType>0:Types.0:DataTypes.0:BaseDataType.0:Boolean</DataType>
   <NodeId>i=11024</NodeId>
   <BrowseName>bool array</BrowseName>
   <DisplayName>bool array</DisplayName>
   <AccessLevel>0x03</AccessLevel>
   <ValueRank>1</ValueRank>
   <AccessLevel>0x03</AccessLevel>
   <ArrayOffset>0</ArrayOffset>
   <ArrayLength>10</ArrayLength>
</Variable>
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 instantiated nodes. The "Operator" role can also write.

```xml
<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.

```xml
<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:

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

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**

```
[OpcUs]
[OpcUa.Trace]
TraceMask = 15
TraceFileSizeKbMax = 4096
```
12 Software units

Within the system it is possible to assemble user-specific software or additional independent extensions ("features") as a software unit and mount them in the u-create studio as an optional package when creating a new target ("Create target").

The software unit must be created for the desired target system and stored as its own directory in the path of the Software Service service. To do this, click on the icon of the Software Service in the right area of the task bar and open the location via the context menu "Open Software Service Path".

The directory name of the software unit must be composed as follows:

\texttt{<name>_<version>_TARGET\_platform>}

\textbf{Example:} \texttt{softwareunitdemo_1.0_x86}

12.1 Creating a software unit

Each software unit must contain at least one repository and one description file (\texttt{meta.xml}). This is the only way to use the automatic installation and uninstallation process of Debian-based control software.

\begin{center}
\includegraphics[width=0.5\textwidth]{structure.png}
\end{center}

\textbf{Fig. 12-11: Structure}

\textbf{Description file "meta.xml"}

The file is the description of the software unit. This contains general information (such as name, version, description,...).

We recommend that you create a separate directory in the software unit directory for each repository within a software unit.
The name under the tag <name> is specified as package name in the "Optional Packages" dialog. The description under the tag <description> is displayed as additional text below.

The installable repository is specified under the tag <data>.

### Repository

A repository must contain installable packages (e.g. *.deb files) and make the sources main, contrib and non-free available.

Each software unit may only contain exactly one repository for exactly one target system (e.g. Repository target="Debian" type="Full").

A list of the packages contained in the repository can be defined. Each package has a unique identifier (Identifier string and InstallPackage string) that can be used by installation programs.

Each repository must contain a meta-packet that has exactly the same Identifier string as the entire software unit.

Normally, a Debian package must be signed with a "gpg" key. By default, the Debian-based control system only contains the public "gpg" key of the Weidmüller software units. If a software unit is installed that is signed with a user-specific "gpg" key, an authentication warning is written to the installation log file.

---

**Information**

*If only individual files are to be copied to the controller, it is recommended that they are also included in a package. This is the only way to ensure clean management (installation, uninstallation, update).*

---

### MD5 checksum

We recommend that you generate an MD5 checksum for repositories. This must be entered in the meta.xml file.

---

**Information**

*The checksum must consist of lowercase letters.*

---

### 12.2 Example meta.xml

This chapter shows an example description file meta.xml.

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<SoftwareUnit>
  <DateTime>2017-09-25 10:48:20</DateTime>
  <Name>
    <DefaultString>Software unit demo</DefaultString>
    <LocalizedString culture="de">Software Unit Demo</LocalizedString>
    <LocalizedString culture="en">Software unit demo</LocalizedString>
  </Name>
</SoftwareUnit>
```
Software units

<Name>
<Description>
<DefaultString>Sample for custom software unit</DefaultString>
<LocalizedString culture="de">Example for customer-specific Software unit</LocalizedString>
<LocalizedString culture="en">Sample for custom software unit</LocalizedString>
</Description>
<Identifier>softwareunitdemo</Identifier>
-Version>1.0</Version>
<Target>Debian</Target>
<Platform>x86</Platform>
<Capabilities>
<Provides>
<Feature version="1.0" identifier="softwareunitdemo" />
</Provides>
<Requires> </Requires>
</Capabilities>
<Data>
<Repositories>
<Repository type="Full">
<Path>repository</Path>
<AvailablePackages>
<Package>
{Name>
<DefaultString>Software unit demo</DefaultString>
<LocalizedString culture="de">Software Unit Demo</LocalizedString>
<LocalizedString culture="en">Software unit demo</LocalizedString>
</Name>
<Description>
<DefaultString>My Software unit 1</DefaultString>
<LocalizedString culture="de">Meine Software Unit 1</LocalizedString>
<LocalizedString culture="en">My software unit 1</LocalizedString>
</Description>
<InstallPackage>swunitdemopacket1</InstallPackage>
<Identifier>softwareunitdemo</Identifier>
</Package>
</AvailablePackages>
<Checksum type="MD5">5342EA09691F4764DD1A02693422C469</Checksum>
<ChecksumFile>repository/dists/jessie-kebian/Release</ChecksumFile>
</Repository>
</Repositories>
</Data>
</SoftwareUnit>
13 **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.

### 13.1 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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Data Type</th>
<th>Data Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>REAL</td>
<td>LREAL</td>
<td>BYTE</td>
</tr>
<tr>
<td>ENUM</td>
<td>WORD</td>
<td>DWORD</td>
<td>LWORD</td>
</tr>
<tr>
<td>SINT8</td>
<td>SINT16</td>
<td>SINT32</td>
<td>SINT64</td>
</tr>
<tr>
<td>ENUM_SINT8</td>
<td>ENUM_SINT16</td>
<td>ENUM_SINT32</td>
<td>ENUM_SINT64</td>
</tr>
<tr>
<td>UINT8</td>
<td>UINT16</td>
<td>UINT32</td>
<td>UINT64</td>
</tr>
<tr>
<td>ENUM_UINT8</td>
<td>ENUM_UINT16</td>
<td>ENUM_UINT32</td>
<td>ENUM_UINT64</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE64us</td>
<td>TIME</td>
<td>TIME64us</td>
</tr>
<tr>
<td>DT</td>
<td>DT64us</td>
<td>TOD</td>
<td>TOD64us</td>
</tr>
</tbody>
</table>
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 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 the CompactFlash (EMMC). 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.

### 13.2 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:

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recording</td>
<td>A recording is carried out.</td>
</tr>
</tbody>
</table>
**State** | **Description**
--- | ---
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 \( \text{prodCnt} \) and a reduction of the storage by a factor of 2, only the values that are written in bold are stored.

<table>
<thead>
<tr>
<th>prod-Cnt</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>14</td>
<td>15</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>tmp</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>61</td>
<td>61</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

**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)
**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.

13.3 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

\[\text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample}\]

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

\[\text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample} \quad \text{Sample}\]

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
Example

Start position: Newest sample

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.

13.4 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.*

### 13.5 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.

### 13.6 Size limitations

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

<table>
<thead>
<tr>
<th>Max. buffer capacity</th>
<th>Max. number of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>524,288</td>
<td>1</td>
</tr>
<tr>
<td>262,144</td>
<td>2</td>
</tr>
<tr>
<td>131,072</td>
<td>4</td>
</tr>
<tr>
<td>65,536</td>
<td>8</td>
</tr>
<tr>
<td>32,768</td>
<td>16</td>
</tr>
<tr>
<td>16,384</td>
<td>32</td>
</tr>
<tr>
<td>8,192</td>
<td>64</td>
</tr>
<tr>
<td>4,096</td>
<td>128</td>
</tr>
</tbody>
</table>
### 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.

### 13.7 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:

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

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).

13.7.1 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.

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.

```c
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);
}
```

/
Data recorder

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;
}

13.7.2 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 fur-
thermore 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 de-
termination.

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;
}

13.7.3 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 \em 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 \em 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 \em 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 \em temperature and \em 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 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);

    strncpy(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;
    strncpy(trig.startCond.var, varPath, sizeof trig.startCond.var - 1);
    trig.stopCond.op = DataRecApiTriggerOpNegSlope;
    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;
}

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

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

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

    rc = DataRecApiStartRecording(sPlId);
    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(sPlId);
    ReturnIfNot(rc == DataRecApiResultOk);
}

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

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

```c
/* 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;
}
```
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`)!

13.7.5 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`).

```c
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(sPlId, 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:

```c
static const char *TimeStamp2DateStr(uint64_t timeStamp) {
    static char sDateStr[80];
    struct tm *p;

    /* seconds and μs since 1970-01-01 */
    time_t secs, usecs;
    struct tm *p;
    secs = (int)(timeStamp / 1000000);
    usecs = (int)(timeStamp % 1000000);

    p = localtime(&secs);
    if (p != 0) {
        strftime(sDateStr, sizeof sDateStr, "%04u-%02u-%02u %02u:%02u:%02u.%03u", p,
                 (unsigned)usecs);
    } else {  // Placeholder for unknown time
        strftime(sDateStr, sizeof sDateStr, "xxxx-xx-xx xx:xx:xx.xxx");
    }
    return sDateStr;
}
```

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

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

**Information**

*The delivery DVD includes the file "Provider_Parameter_Overview.xls*. There variables, which are shown in the variable monitor, are described with corresponding units, data types and default values.*

14.1 Control diagnosis

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

**PWR**

<table>
<thead>
<tr>
<th>Indication</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dark</td>
<td>No voltage supply</td>
</tr>
<tr>
<td>Green</td>
<td>Device running</td>
</tr>
</tbody>
</table>

**SF**

<table>
<thead>
<tr>
<th>Indication</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dark</td>
<td>No voltage supply or no error</td>
</tr>
<tr>
<td>Red</td>
<td>Severe system error (e.g. Fatal Error)</td>
</tr>
</tbody>
</table>

**RUN**

<table>
<thead>
<tr>
<th>Indication</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dark</td>
<td>No voltage supply</td>
</tr>
<tr>
<td>Green</td>
<td>Application is processed or setup finished</td>
</tr>
<tr>
<td>Flashing green</td>
<td>Application is stopped</td>
</tr>
<tr>
<td>Yellow</td>
<td>Ready for operation</td>
</tr>
<tr>
<td>Flashing yellow</td>
<td>Setup is executed</td>
</tr>
</tbody>
</table>

**Link/Activity LED**

<table>
<thead>
<tr>
<th>Indication</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dark</td>
<td>No connections</td>
</tr>
<tr>
<td>Flashing green/yellow</td>
<td>Data is transferred</td>
</tr>
<tr>
<td>Green</td>
<td>EtherCAT connection established (100 MBit/s, Full Duplex)</td>
</tr>
</tbody>
</table>

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)).
14.1.1 Error codes

Depending on the version of the CPU module, different error messages can be displayed.

14.2 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.

14.2.1 Status report

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

• Directly on the CPU module
• Via Service-App (DevAdmin)

**Triggering the status report directly on CPU module**

The procedure to trigger the status report directly on the CPU module varies depending on the model. Detailed information can be found in the project engineering manual of the respective module.

**Triggering the status report via Service-App**

See "Device Administration (DevAdmin)".

14.2.2 Access to flash storage medium

The access to the flash storage medium can only happen via a SSH encrypted SFTP connection. Therefore a FTP program which is installed on the PC is needed. FTP programs are offered for download in the internet.

To establish a connection to the control with the FTP program the IP address of the control with the port number must be input into the FTP program.

In addition the following data which must also be inserted into the FTP program is needed for the connection establishment:

• Login data
• SSH key

For a detailed description see the help of the FTP program.

**Information**

_The selected FTP program must support an authentication via SSH encryption._
Login Data

The following user exists on the control by default and can be used for the connection:

- service
- Admin

**Information**

*The login data cannot be changed.*

SSH key

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

**Changing the SSH key pair**

To generate a new SSH key pair a special program is needed. This program can be downloaded in the internet whereat the program "PuTTYgen" is recommended.

To change the SSH key with the use of "PuTTYgen" proceed as follows:

1) Start "PuTTYgen" on PC.

![PuTTYgen](image)

2) Select the type Type "SSH-2 RSA" (selected by default) in the lower area of the dialog.

3) Generate key via "Generate".
4) Save the private key via "Save private key" ([username]_key.ppk) to a directory on the PC.
5) Create a new text file named authorized_keys to a directory on the PC.
6) Mark and copy the whole text in the text field in the upper area of the download below "Public key for pasting into OpenSSH authorized_keys file:"

7) Copy the text into the file authorized_keys.
8) Save and close the file.

The SSH key pair was generated.

The public key must be downloaded to the control. This is done differently for the existing users.

**User "service"**

To download the public key for the user "service" proceed as follows:

1) Start the FTP program and connect to the control.
2) Switch to the direction ssh-key and download the file authorized_keys from the PC to the control (existing file is overwritten).
3) Restart the control.

The SSH key has been changed. The new private key must be given when connecting now via SFTP connection to the control.

**User "Admin"**

To download the public key for the user "service" proceed as follows:

1) Right mouse click on icon "Software Service" in the taskbar of the PC.
2) Open path via "Open Software Service Path".
3) Copy the file authorized_keys in the corresponding directory:

<table>
<thead>
<tr>
<th>Taget directory</th>
<th>Target device</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\data\&lt;dsarmxilinxkebwm_VersionNr&gt;\config\key\admin</code></td>
<td>UC20-SL2000-OLAC-EC</td>
</tr>
<tr>
<td><code>\data\&lt;dsx86genkeb_VersionNr&gt;\config\key\admin</code></td>
<td>KEBA control</td>
</tr>
</tbody>
</table>

The new public key is now stored and will be automatically downloaded the next time "Create Target" (via .u-create studio) is executed. Then the SSH key has been changed. The new private key must be given when connecting now via SFTP connection to the control.
# Directories on the flash storage medium

The user is able to read and write in the following directories, but is not allowed to execute files:

<table>
<thead>
<tr>
<th>Verzeichnis</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>appldisk</td>
<td>Root directory for applications (Visualization application, IED, ...)</td>
</tr>
<tr>
<td>masterdisk</td>
<td>Root directory for base system</td>
</tr>
<tr>
<td>ssh-keys</td>
<td>Directory for the public key which can be changed</td>
</tr>
<tr>
<td>var</td>
<td>In this directory sub-directories for state report and crashreport are located</td>
</tr>
<tr>
<td>workspace</td>
<td>Root directory for files which are created during operation</td>
</tr>
</tbody>
</table>
15 Technical data

Detailed information concerning the technical data can be found in the respective project engineering manuals.
16  EC Directives and Standards

The details of the EC Directives and standards observed can be found in the relevant project engineering manuals.
17 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 controller and executed.

17.1 Creating a new project

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

![u-create studio - New project](image)

Fig. 17-13: 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:
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...`.
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:

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.
17.2 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...**:

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

\[ x := x + 1; \]

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:
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.

![Task configuration](image)

**Fig. 17-22: 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.

![Appending POU to the task](image)

**Fig. 17-23: 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.
17.3 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.

17.4 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**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Server</td>
<td>localhost</td>
</tr>
<tr>
<td>System</td>
<td>System version which should be installed on the control or the active static operating panel</td>
</tr>
</tbody>
</table>

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:

![Create Target dialog](image)

Fig. 17-24: 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 Service 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.

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.

17.5 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:

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.

17.6 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):

![Authentication dialog](image)

Fig. 17-26: 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.

![Selective Download to Device](image)

Fig. 17-27: 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 soft-
ware 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.

17.7 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. 17-28: Started sample project](image-url)
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:

<table>
<thead>
<tr>
<th>Net class</th>
<th>Net mask</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>255.0.0.0</td>
<td>Large network</td>
</tr>
<tr>
<td>B</td>
<td>255.255.0.0</td>
<td>Medium-sized network</td>
</tr>
<tr>
<td>C</td>
<td>255.255.255.0</td>
<td>Small network with a maximum of 254 participants</td>
</tr>
</tbody>
</table>
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.
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++. Therefore a template with a predefinded interface for data exchange exists in u-create studio C++. It is recommended 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 conversion.

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 in u-create studio.

19.1 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

The creation of this example requires basic knowledge in dealing with u-create studio and u-create studio C++.

19.2 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

### 19.3 Creation of IEC function

As a first step start u-create studio and open an executable project.

To call a C function within an IEC function the library `K_BinFramework` must be added in the library manager.

Create a function with the name `MyCFunction` in the project tree of the IEC application via **Add object ► POU...** and open it in the working area.

In the created function the following interface and the necessary local variables for data exchange with the C function are defined:

**Inputs**

```plaintext
VAR_INPUT
    in_dword : DWORD;
    in_string : STRING;
    in_real : REAL;
END_VAR
```

**Outputs**

```plaintext
VAR_OUTPUT
    out_real : REAL;
    out_string : STRING;
```

**Local variables**

```plaintext
VAR
    rv: DINT; (*Return value of the C function*)
    pInput : ARRAY [1..3] OF CFuncParam ; (*Input parameter C function*)
    pOutput : ARRAY [1..2] OF CFuncParam ; (*Output parameter C function*)
    pInst : ARRAY [1..1] OF CFuncParam ; (*Additional parameter C function*)
END_VAR
```

Additionally create a global variable list in the project tree under **IEC_C_Call ► <control> ► PLC logic ► Application**. This is done via context menu **Add object**. In this a global variable (for performance reasons) and a handle of the C function are defined.

**Global variables**

```plaintext
VAR_GLOBAL
    MyCFunctionHdl : DWORD := 0;
    inst1 : DWORD;
END_VAR
```
In the code area of the function first the handle to the desired C function is fetched. This happens via function `GetCFuncHdl`. The transferred parameters of the function are the name of the C function and the path in which the `.so` file is stored later on the control.

```
IF MyCFunctionHdl = 0 THEN
    MyCFunctionHdl := GetCFuncHdl('func1', '/appldisk/application/control/ccontrol/libMyIEC_C_Call.so');
END_IF;
```

To access the parameters of the IEC function in the C function the address to the memory area must be written into the predefined transfer structure.

```
pInput[1].pData := ADR(in_dword);
pInput[2].pData := ADR(in_string);
pInput[3].pData := ADR(in_real);
pOutput[1].pData := ADR(out_real);
pOutput[2].pData := ADR(out_string);
pInst[1].pData := ADR(inst_global);
```

Then the C function is called by the function `CallCFunc`, if the handle is valid. The return value is stored in variable `rv`.

```
IF MyCFunctionHdl > 0 THEN
    rv := CallCFunc(Hdl := MyCFunctionHdl, pInputParam := ADR(pInput), pOutputParam := ADR(pOutput), pInstParam := ADR(pInst));
END_IF
```

The return value of the C function is returned via the return value of the IEC function.

```
MyCFunction := rv;
```

The function to call the C function has been created and can be used now in the IEC application.

19.4 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>myProject</td>
</tr>
<tr>
<td>Location</td>
<td>C:\myProjects</td>
</tr>
<tr>
<td>Target</td>
<td>● UC20-SL2000: Control PLC V&lt;VersionNo&gt; ARMHF Linux</td>
</tr>
<tr>
<td></td>
<td>● KEBA control: Control PLC V&lt;VersionNo&gt; X86 Linux</td>
</tr>
<tr>
<td>Project Type</td>
<td>CoDeSys IEC Functions in C</td>
</tr>
</tbody>
</table>

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:

```c
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:

```c
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:

```c
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.

```c
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`:

```c
*(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`:

```c
*(in->out_dword) = *(in->in_real) * 3;
```
*(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`.

```c
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.

```c
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 `/appldisk/application/control/ccontrol/` via `Download`.

### 19.5 Calling the IEC function in the IEC application and download

In the IEC application the created IEC function can be used. Therefore a program (e.g. `PLC_PRG`) in u-create studio is extended with the following code:

**Local variables**

```c
VAR
    rv1: DINT;
    res1 : REAL;
    res2 : STRING [106];
END_VAR
```

In the code area the IEC function `MyCFunctor` is called as follows:

```c
rv1 := MyCFunctor( 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

C
C functions .................................................. 114
CAN ............................................................. 22
Configuration
  Control .................................................... 109
Creating project ........................................... 101

D
DevAdmin
  Changing password ...................................... 52
  Crashreport ................................................. 50
  Date ............................................................ 52
  Device Information ....................................... 49
  Device State ............................................... 50
  Interface Ethernet ....................................... 51
  Login data ................................................... 48
  Long-term Diagnostic Monitor ......................... 52
  Network Hostname ........................................ 51
  Statereport ................................................. 50
  Time .......................................................... 52

E
EtherCAT ...................................................... 21
Ethernet
  Adressing .................................................. 112

F
Fast control .................................................. 33
Firmware
  Installing .................................................. 30

I
Interfaces
  CAN .......................................................... 22
  EtherCAT .................................................. 21
  Modbus TCP/IP ........................................... 22
  IP address .................................................. 18

M
Modbus TCP/IP ............................................. 22

N
Network address ............................................ 18

P
Programming languages ................................... 23

S
Service PC
  Options for connections .................................. 19
Status report ............................................... 95

T
Tools
  KeStudio .................................................. 23
  Scope ........................................................ 25
  u-create studio C++ ....................................... 24