1 General information

OPC Unified Architecture (OPC UA) is a software interface for an industrial communication protocol and data model for communication between end devices (e.g. PLCs, industrial PCs, process control systems) of different manufacturers according to the client-server principle. OPC UA is used for the reliable and manufacturer-independent transport of data (raw data and pre-processed information from the sensor level to the PLC / control computer). With the help of OPC UA, any type of information can be made available anytime and anywhere.

Properties of OPC UA
- Communication via TCP/IP and Internet
- Communication across firewall boundaries
- Easy configuration and maintenance
- Secure, encryptable and uses certificates
- All OPC UA information (e.g. data) available in a namespace
- Can be used on different operating systems and with different programming languages
2 Technical description

2.1 Order data

<table>
<thead>
<tr>
<th>Model number</th>
<th>Short description</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>X20BC008U</td>
<td>X20 bus controller, 1 OPC UA Ethernet interface, integrated 2-port switch, 2x RJ45, order bus base, power supply module and terminal block separately.</td>
<td></td>
</tr>
</tbody>
</table>

**Required accessories**

<table>
<thead>
<tr>
<th>Model number</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X20BB80</td>
<td>X20 bus base, for X20 base module (BC, HB, etc.) and X20 power supply module, X20 end plates (left and right) X20AC05SR1 included</td>
</tr>
<tr>
<td>X20PS9400</td>
<td>X20 power supply module, for bus controller and internal I/O power supply, X2X Link power supply</td>
</tr>
<tr>
<td>X20PS9402</td>
<td>X20 power supply module, for bus controller and internal I/O power supply, X2X Link power supply, supply not electrically isolated</td>
</tr>
</tbody>
</table>

**Terminal blocks**

<table>
<thead>
<tr>
<th>Model number</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X20TB12</td>
<td>X20 terminal block, 12-pin, 24 VDC keyed</td>
</tr>
</tbody>
</table>

Table 1: X20BC008U - Order data
2.2 Technical data

<table>
<thead>
<tr>
<th>Model number</th>
<th>X20BC008U</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short description</td>
<td>OPC UA server</td>
</tr>
<tr>
<td>General information</td>
<td></td>
</tr>
<tr>
<td>B&amp;R ID code</td>
<td>0xEAFB</td>
</tr>
<tr>
<td>Status indicators</td>
<td>Module status, bus function</td>
</tr>
<tr>
<td>Diagnostics</td>
<td></td>
</tr>
<tr>
<td>Module status</td>
<td>Yes, using status LED and software</td>
</tr>
<tr>
<td>Bus function</td>
<td>Yes, using status LED and software</td>
</tr>
<tr>
<td>Power consumption</td>
<td></td>
</tr>
<tr>
<td>Bus</td>
<td>2.5 W</td>
</tr>
<tr>
<td>Additional power dissipation caused by actuators (resistive) [W]</td>
<td>-</td>
</tr>
<tr>
<td>Certifications</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>Yes</td>
</tr>
<tr>
<td>EAC</td>
<td>Yes</td>
</tr>
<tr>
<td>UL</td>
<td>cULus E115267</td>
</tr>
<tr>
<td>ATEX</td>
<td>Zone 2, I 3G Ex nA nC IIA T5 Gc</td>
</tr>
<tr>
<td></td>
<td>IP20, Ta (see X20 user's manual)</td>
</tr>
<tr>
<td></td>
<td>FTZÚ 09 ATEX 0083X</td>
</tr>
<tr>
<td>Interfaces</td>
<td></td>
</tr>
<tr>
<td>Fieldbus</td>
<td>OPC UA server</td>
</tr>
<tr>
<td>Variant</td>
<td>2x shielded RJ45 (switch)</td>
</tr>
<tr>
<td>Line length</td>
<td>Max. 100 m between 2 stations (segment length)</td>
</tr>
<tr>
<td>Transfer rate</td>
<td>100 Mbit/s</td>
</tr>
<tr>
<td>Transfer</td>
<td></td>
</tr>
<tr>
<td>Physical layer</td>
<td>100BASE-TX</td>
</tr>
<tr>
<td>Half-duplex</td>
<td>Yes</td>
</tr>
<tr>
<td>Full-duplex</td>
<td>Yes</td>
</tr>
<tr>
<td>Autonegotiation</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-MDI / MDIX</td>
<td>Yes</td>
</tr>
<tr>
<td>Min. cycle time 1)</td>
<td></td>
</tr>
<tr>
<td>Fieldbus</td>
<td>10 ms</td>
</tr>
<tr>
<td>X2X Link</td>
<td>2 ms</td>
</tr>
<tr>
<td>Synchronization between bus systems possible</td>
<td>Yes</td>
</tr>
<tr>
<td>Electrical properties</td>
<td>OPC UA isolated from bus and I/O</td>
</tr>
<tr>
<td>Operating conditions</td>
<td></td>
</tr>
<tr>
<td>Mounting orientation</td>
<td>Horizontal: Yes, Vertical: Yes</td>
</tr>
<tr>
<td>Installation elevation above sea level</td>
<td>0 to 2000 m: No limitations, &gt;2000 m: Reduction of ambient temperature by 0.5°C per 100 m</td>
</tr>
<tr>
<td>Temperature</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td></td>
</tr>
<tr>
<td>Horizontal mounting orientation</td>
<td>-25 to 60°C</td>
</tr>
<tr>
<td>Vertical mounting orientation</td>
<td>-25 to 50°C</td>
</tr>
<tr>
<td>Derating</td>
<td>-</td>
</tr>
<tr>
<td>Storage</td>
<td>-40 to 85°C</td>
</tr>
<tr>
<td>Transport</td>
<td>-40 to 85°C</td>
</tr>
<tr>
<td>Relative humidity</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>5 to 95%, non-condensing</td>
</tr>
<tr>
<td>Storage</td>
<td>5 to 95%, non-condensing</td>
</tr>
<tr>
<td>Transport</td>
<td>5 to 95%, non-condensing</td>
</tr>
<tr>
<td>Mechanical properties</td>
<td>Order terminal block 1x X20TB12 separately, Order 1x power supply module X20PS9400 or X20PS9402 separately, Order 1x bus base X20BB80 separately</td>
</tr>
<tr>
<td>Spacing 2)</td>
<td>37.5 ( \pm 0.2 ) mm</td>
</tr>
</tbody>
</table>

Table 2: X20BC008U - Technical data

1) The minimum cycle time defines how far the bus cycle can be reduced without communication errors occurring.
2) Spacing is based on the width of bus base X20BB80. In addition, power supply module X20PS9400 or X20PS9402 is always required for the bus controller.
2.3 LED status indicators

The following table lists the LED status indicators available on the bus controller. Exact blink times are specified in the timing diagram in the next section.

<table>
<thead>
<tr>
<th>Figure</th>
<th>LED</th>
<th>Color</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS(1)</td>
<td>Green</td>
<td>Off</td>
<td>X2X halted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 pulses</td>
<td>Initializing X2X network</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>On(2)</td>
<td>Normal operation, X2X active</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Red</td>
<td>1 pulse</td>
<td>One or more modules missing since startup</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 pulses</td>
<td>I/O module missing that was already active</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 pulses</td>
<td>Mismatch of at least 1 I/O module with configuration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 pulses</td>
<td>Watchdog timer expired</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>On</td>
<td>Corrupt configuration file in flash memory or hardware fault</td>
<td></td>
</tr>
<tr>
<td>CS(3)</td>
<td>Green</td>
<td>Off</td>
<td>No active OPC UA sessions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 pulse</td>
<td>Waiting for DHCP server(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 pulses</td>
<td>Waiting for time synchronization(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>On</td>
<td>At least 1 connection is active</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Red</td>
<td>1 pulse</td>
<td>Waiting for DHCP server(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 pulses</td>
<td>Waiting for time synchronization(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 pulses</td>
<td>Error mode OutputKeepAlive is active</td>
<td></td>
</tr>
<tr>
<td>L/A IFx</td>
<td>Green</td>
<td>Off</td>
<td>Indicates that no physical Ethernet connection exists</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blinking</td>
<td>Indicates an established connection (link), but no communication is taking place</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>On</td>
<td>Indicates an established connection (link), but no communication is taking place</td>
<td></td>
</tr>
</tbody>
</table>

1) This LED is a green/red dual LED.
2) The red LED can be superimposed over the green LED, which results in a mixed color.
3) The waiting time is indicated by green pulses in the first 10 seconds of the search or synchronization; red pulses then follow.

LED status indicators - Blink times

![Blink times diagram]

2.4 Ethernet interface

For information about wiring X20 modules with an Ethernet interface, see section "Mechanical and electrical configuration - Wiring guidelines for X20 modules with Ethernet cables" of the X20 user's manual.

![Ethernet interface diagram]

<table>
<thead>
<tr>
<th>Interface</th>
<th>Pin</th>
<th>Ethernet</th>
<th>Pinout</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ45, shielded</td>
<td>1</td>
<td>RXD</td>
<td>Receive data</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>RXD1</td>
<td>Receive data1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>TXD</td>
<td>Transmit data</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>TXD1</td>
<td>Transmit data1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Termination</td>
<td></td>
</tr>
</tbody>
</table>
2.5 Network address switches

The network address switches have multiple functions:

- Changing the default IP address (range 0x01 to 0x7F)
- Enables operation with a DHCP server (in the range 0x80 to 0xEF)
- Initializes the communication parameters with their default values (0xFF)

For an overview of network address switch functions, see “Overview of network address switch functionality” on page 7.

**Information:**

- It is important to make sure that 2 bus controllers are not being operated in the same network with the same network address switch settings, with the exception of value 0x00.
- For all switch positions except 0x00, the IP address configured in the bus controller is not or only partially used (range 0x01 to 0x7F). The IP address actually used is indicated in OPC UA object "CurrentIP-Address" after startup.
- Changes to the network address switches are only applied after a restart.

2.6 Overview of network address switch functionality

The bus controller has 2 address switches. The bus controller can be set to different operating modes using certain, pre-defined switch positions.

<table>
<thead>
<tr>
<th>Switch position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>All parameters are initialized from flash memory. The bus controller is started with its default values if valid flash data is not present. In this case the configuration is identical to switch position 0xFF.</td>
</tr>
</tbody>
</table>
| 0x01 - 0x7F     | Parameters are read from flash memory. The last position of the IP address saved in flash memory is changed to the address switch value, however. Example  
• IP address stored in flash memory: 192.168.1.1  
• Switch position: 0x20 (decimal 32)  
• Resulting IP address: 192.168.1.32 |
| 0x80 - 0xEF     | In this range, the bus controller operates in DHCP mode. A hostname is generated according to the setting of the address switch. The generated hostname is composed of 3 elements: →“br” + “opc” + Address switch value (3 decimal positions)  
Example  
If the address switch value is 0xD7 (dec. 215), for example, then hostname “bropc215” will be generated. |
| 0xF0            | See “Restoring to factory settings” on page 10. |
| 0xF1 - 0xFE     | Reserved, switch position not permitted |
| 0xFF            | All parameters are set to their default values. Parameters in flash memory are not overwritten. The following default values are used:  
• IP address: 192.168.1.1  
• Subnet mask: 255.255.255.0  
• Default gateway: 192.168.1.254  
• DNS server: 192.168.1.254  
• OPC UA server port: 4840 |
2.7 Configures network settings

A network connection and assigned IP address are required to communicate with the bus controller. The following variants are possible for assigning the IP address:

- Static IP address
- Address assignment by a DHCP server
- Use of OPC UA objects

Default parameters since firmware version 1.01

- IP address: 192.168.1.1
- Subnet mask: 255.255.255.0
- Default gateway: 192.168.1.254
- DNS server: 192.168.1.254
- OPC UA server port: 4840

2.7.1 Connecting to the bus controller via Ethernet

The connection between the network and bus controller can be made in the following way:

- Direct connection via patch cable between the PC network connection and bus controller.
- Over an Ethernet network. If necessary, multiple bus controllers can be connected to the network at the same time.

Straight-through or crossover Ethernet cables can be used. Ethernet interface IF1 or IF2 is permitted to be used as the slot on the bus controller.

Since the default subnet mask of the bus controller is 255.255.255.0, the first 3 bytes of the IP address for the PC must match that of the bus controller.

Example

The bus controller has the default IP address of 192.168.1.1. In this case, the PC must be set to address 192.168.1.xxx, with xxx between 2 and 254.

The B&R OPC UA bus controller can be addressed in 2 ways:

- Via its IP address
- Via its hostname

The IP address of the controller can be altered using its network address switches. The (configured) IP address and port number stored in the controller's flash memory are used in position 0x00.

If the network address switch is set to 0xFF, the controller is assigned IP address 192.168.1.1 on restart.

For additional details about address switches, see "Overview of network address switch functionality" on page 7.

2.7.2 Automatic IP assignment by a DHCP server

If a network address switch setting between 0x80 and 0xEF is configured, the bus controller will attempt to request an IP address from the DHCP server. The assigned IP address can be queried with command "ping" together with the hostname. The bus controller registers the hostname on the DHCP server, which should forward it to a DNS server.

Example

The hostname (DNS name) is made up of 3 elements:
"br" + "opc" + Address switch value (3 decimal places)
This means, for example, that the following hostname is generated for address switch setting 0xD7 (dec. 215): "bropc215".

If no DNS service is available in the network, access is also possible via the two NetBIOS names of the bus controller. The secondary NetBIOS name is identical to the hostname. If the address switches are set to 0x00, it is identical to the primary NetBIOS name.

The bus controller can only be reached via its NetBIOS name if no other routers or gateways are in the way.
2.7.3 Setting network parameters manually

Operation on a DHCP server can also be configured using by setting object "X20BC008U / Configuration / Network / EnableDHCP" to TRUE. With this setting, the hostname must be assigned by the DNS server.

**Information:**

Changes to attributes in the TCP/IP object are only saved to flash memory after OPC UA object ApplyChanges or ApplyAllConfigurations is called. They are applied after the bus controller has been restarted unless the network address switch forces another setting.

2.7.4 Changing the IP address with the network address switches

The address switches can be used to change the last byte in the IP address configured on the bus controller. The IP address saved in flash memory is not changed. If the address switches are set to 0x00, the bus controller applies the IP address last saved to flash memory.

Switch positions between 0x01 and 0x7F cause the last position of the IP address (the lowest byte) to be overwritten by the value of the address switch. This provides the user a quick and easy way to address a large number of bus controllers. In short, an IP address between 192.168.1.1 and 192.168.1.127 can be selected for a bus controller using the address switches without requiring any additional software configuration.

2.7.5 Information about NetBIOS names

In addition to the hostname used to register on the DHCP server, the bus controller also has so-called NetBIOS names. These are used to access the bus controller from a PC using its name (as opposed to its IP address). This is only possible if no routers or gateways are in the way, however.

The primary NetBIOS name is always composed of the prefix "br" and the MAC address from the bus controller (see "Automatic IP assignment by a DHCP server" on page 8).

The secondary NetBIOS name corresponds to the primary NetBIOS name at address switch position 0x00. This is necessary because several bus controllers with address switch value 0x00 are permitted to be located in one network segment. In this case, the IP address from flash memory is used.

For all other address switch positions, the secondary NetBIOS name is generated from the network address switch value (as in DHCP mode): "br" + "opc" + Address switch value (3 decimal places).

A hostname defined explicitly by the user will be used for the secondary NetBIOS name regardless of the address switch value.

This makes it possible to access the bus controller with the NetBIOS name configured using the address switches. This is also possible if the controller was not configured for use with a DHCP server (address switch setting between 0x01 and 0x7F).
2.8 Boot behavior and factory settings

2.8.1 Restarting the bus controller

During bus controller or X2X module configuration, a restart of the bus controller may be required for settings to be applied. It is sufficient to perform all configuration activities first and then to perform a single restart, regardless of when the restart was reported as necessary.

Since a required restart cannot be predicted by the user, the configuration status must always also be evaluated during configuration.

The bus controller can be restarted in the following ways:

- Temporarily interrupting the power supply.
- Via bus controller OPC UA object Reboot

2.8.2 Restoring to factory settings

The following procedure is used to reset the bus controller to its default configuration.

1) Set the address switch to F0.
2) Switch on the power supply.
3) If LED "MS" is blinking red, it indicates that the deletion operation can be started.
4) To start the deletion procedure, set the address switch to 00 for 1 second and then set it back to F0.
5) If LED "MS" is solid red, it indicates that deletion is in progress; this lasts approximately 20 seconds. When deletion has completed, LED "MS" blinks red and green alternately.
6) Reset the address switch to the desired value and reboot the bus controller.

**Information:**

If the deletion procedure is interrupted due to a power failure, the operation must be restarted; otherwise, parts of the previous configuration may be retained.

2.8.3 Forcing a boot from the factory default sector

This is necessary if a firmware has been stored in the upgrade sector that operates the watchdog correctly but does not allow an error-free boot process. The bootloader would start the defective firmware, no longer providing a way to perform a subsequent update.

To force a boot from the factory default sector, one of the network address switches must be continuously moved during the actual boot procedure. This is detected by the bootloader, which causes module status LED "MS" to begin flickering red very rapidly. After 1 second passes in which the network address switch is no longer changed, the bus controller restarts using the factory default boot sector and the current value of the network address switches.

The bus controller boots into this mode with limited basic functionality. This allows the establishment of connections, configuration, error analysis and firmware updates.

**Information:**

With bus controllers with a hardware revision <B7, no network connection can be established after such a boot process since an incompatible subnet of 192.168.100.x is set by default in the boot version.
2.9 Web server

The web server integrated in the bus controller in Rev. B0, firmware V.0.09 and later can be used for the firmware update. There are no extensive diagnostic functionalities, however.

The web server can be opened in any browser by specifying the IP address or hostname in the address bar.

**Examples**

http://192.168.1.1

http://bropc128 with an address switch value of 0xEF or 0x80.

2.9.1 Firmware update

Using the Ethernet interface and **configured IP address** of the bus controller, it is possible to connect to the integrated web page in a web browser. The update is started by selecting menu option "Firmware download".

![Firmware Download](image)

**Notes about the firmware update**

The current firmware version, as well as all previous versions, can be downloaded from the website. It is important to note that the firmware filename "60155_1.fw" stored in the ZIP file is identical for all firmware versions. It consists of the module ID (ModuleID=60155) and hardware variant (_1)).

To better differentiate between firmware versions, the version number is included in the ZIP filename in version 1.18 and later.

**Information:**

The X20BC008U does not support automated firmware updates via hardware upgrade. The firmware update must be carried out manually by the user through downloading via the web server.
2.10 Supported OPC UA server facets

The bus controller supports the following server facets:

Core server facet
- Address space base
- Base info server capabilities
- Base info ValueAsText
- Discovery get endpoints
- Session general service behavior
- Session base
- Session minimum 5
- View basic
- View TranslateBrowsePath
- View RegisterNodes
- View minimum continuation point (the number of continuation points is unlimited)
- Attribute read
- Attribute write values

Data access
- Monitor basic
- Monitor value change
- Monitor items 1000
- Monitor QueueSize 100
- Subscription basic
- Subscription minimum 10

Generic features
- Address space method
- Method call

Transport
- UA-TCP UA-SC UA-Binary (see “Protocols” on page 15)

2.11 Supported modules

The following table shows all of the I/O modules that are stored in the bus controller's database. See "X2X Link and I/O modules" on page 40.

<table>
<thead>
<tr>
<th>Model number</th>
<th>Starting with firmware version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X20AI2222</td>
<td>V1.13</td>
<td>X20 analog input module, 2 inputs, ±10 V, 13-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20AI2322</td>
<td>V1.13</td>
<td>X20 analog input module, 2 inputs, 0 to 20 mA / 4 to 20 mA, 12-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20AI2622</td>
<td>V1.00</td>
<td>X20 analog input module, 2 inputs, ±10 V or 0 to 20 mA / 4 to 20 mA, 13-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20AI4222</td>
<td>V1.13</td>
<td>X20 analog input module, 4 inputs, ±10 V, 13-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20AI4322</td>
<td>V1.13</td>
<td>X20 analog input module, 4 inputs, 0 to 20 mA / 4 to 20 mA, 12-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20(c)AI4622</td>
<td>Coated module V1.18</td>
<td>X20 analog input module, 4 inputs, ±10 V or 0 to 20 mA / 4 to 20 mA, 13-bit converter resolution, configurable input filter</td>
</tr>
<tr>
<td>X20AI8221</td>
<td>V1.13</td>
<td>X20 analog input module, 8 inputs, ±10 V, 13-bit converter resolution</td>
</tr>
<tr>
<td>X20AI8321</td>
<td>V1.13</td>
<td>X20 analog input module, 8 inputs, 0 to 20 mA, 12-bit converter resolution</td>
</tr>
<tr>
<td>X20AO2622</td>
<td>V1.00</td>
<td>X20 analog output module, 2 outputs, 10 V or 0 to 20 mA / 4 to 20 mA, 13 bit converter resolution</td>
</tr>
<tr>
<td>X20(c)AO4622</td>
<td>Coated module V1.18</td>
<td>X20 analog output module, 4 outputs, 10 V or 0 to 20 mA / 4 to 20 mA, 13 bit converter resolution</td>
</tr>
<tr>
<td>X20AP3111</td>
<td>V1.13</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 20 mA AC, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
<tr>
<td>X20(c)AP3121</td>
<td>Coated module V1.18</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 1 A AC, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
<tr>
<td>X20AP3122</td>
<td>V1.13</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 1 A AC, groundable, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
<tr>
<td>X20(c)AP3131</td>
<td>Coated module V1.18</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 5 A AC, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
<tr>
<td>X20AP3132</td>
<td>V1.13</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 5 A AC, groundable, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
<tr>
<td>X20AP3161</td>
<td>V1.13</td>
<td>X20 energy metering module, 3 analog inputs, 480 VAC, 50/60 Hz, 4 analog inputs, 333 mV AC, calculates effective, reactive and apparent power/energy, calculates RMS values, 240 V keyed, NetTime function</td>
</tr>
</tbody>
</table>
## 2.12 Module behavior in the event of connection loss

**Caution!**

If the OPC UA connection between the client and bus controller is lost, the current module states are frozen. Outputs are not reset.
3 Basic information

3.1 Connection management

Definition
Establishing a connection refers to the process of connecting 2 network stations, e.g. between a client and server. As soon as a valid connection has been established, this is referred to as the active "connection" or "session".

Closing a connection refers to the disconnection or controlled ending of an active connection.

Connection management as a whole comprises all tasks related to establishing and closing connections and their error handling.

Establishing a connection to the OPC UA bus controller

One of the ways the connection from the client (CPU) to the OPC UA bus controller can be established is using function block `UA_Connect()` in library "AsOpcUa". The function block automatically tries to maintain this connection. The connection remains active until `UA_Disconnect()` is carried out or the connection is disconnected mechanically, e.g. by disconnecting the connection cable.

Please note the following:

• Security functionalities are not yet supported. This means that only a connection of type "UserIdentityTokenType := UAUITT_Anonymous" can be carried out.
• All connections established with `UA_Connect()` must be freed up with `UA_Disconnect()`.
• A maximum of 5 parallel connections to the OPC UA bus controller can be established. Active connections that were not closed using `UA_Disconnect()` continue to use one of the available parallel connections.
• "Obsolete" connections are only freed up again with a firmware version <V1.13 after restarting the bus controller or by using `UA_Disconnect()` on the still active connection.
  With a firmware version ≥V1.13, the connection is closed automatically.

Caution!
If the OPC UA connection between the client and bus controller is lost, the current module states are frozen. Outputs are not reset.

3.1.1 Explanations of terms

DHCP server - Dynamic Host Configuration Protocol

Dynamic Host Configuration Protocol (DHCP) is a communication protocol that enables connected clients to be integrated into an existing network without manual configuration. Necessary information such as IP address, subnet mask, gateway, name server (DNS) is automatically assigned, provided that this operation is supported on the client.

By default, DHCP can assign the following settings to the client:

• IP address and subnet mask
• Default gateway
• DNS server, DNS context and DNS tree
• Secondary DNS server

The bus controller generates unique IP addresses for address switch positions 0x01 to 0x7F and uses the hostname stored in flash memory. This is set to "X20BC008U" by default. Operation with a DHCP server is not possible with these switch settings since the IP addresses are generated according to the address switch.

Example
Address switch position 0x02 → generates IP address 192.168.1.2

Operation of a DHCP and DNS server is required for address switch settings 0xB0 to 0x7F. In this case, the hostname is generated according to the address switch position.
DNS - Domain name server
The domain name server (DNS) is a service in IP-based networks. Its main task is to answer requests for name resolution. A name or hostname is translated into a corresponding numeric address (IP address, IPv4 address).

With the OPC UA bus controller, for example, the IPv4 address 192.168.1.5 stored in flash memory is assigned to hostname "broc229".

HOST - Hostname
The hostname is a unique name for a resource in a network. In contrast to an IP address, it is easier to read and remember for people. The hostname is translated into a machine-readable address (IP address) mainly by DNS servers.

With the OPC UA bus controller, a unique hostname is generated for address switch positions 0xB0 to 0xEF; for example, address switch position 0xE5 = 229 results in hostname "broopc229". The subsequent assignment of the hostname to an IP address must take place via a DNS server.

For address switch positions 0x00 to 0x7F, the hostname "X20BC008U" stored in flash memory is used. In this case, the bus controller is distinguished by its unique IP address.

URL - Uniform Resource Locator
URL stands for "Uniform Resource Locator". Resources such as the OPC UA bus controller can be identified and localized via the access method (for example, the network protocol used such as HTTP or FTP) and its location in networks. In common parlance, URLs are often referred to as Internet addresses or web addresses.

IP address - Internet Protocol address
An Internet Protocol address (IP address) is an address based on the Internet Protocol (IP). Today's most common protocol, IPv4 (Internet Protocol version 4), uses 32-bit addresses. A maximum of 4,294,967,296 addresses can be assigned in this network.

IPv4 addresses are usually written in decimal form in 4 blocks, e.g. 192.168.1.229. 1- and 2-digit numbers are not permitted to begin with the number 0. Each block has a range of values from 0 to 255.

In order to be able to uniquely assign public IP addresses worldwide, their assignment is regulated by the Internet Assigned Numbers Authority (IANA).

Private or company networks are marked by the reserved addresses 192.168.xxx.xxx or 10.xxx.xxx.xxx. For the OPC UA bus controller, for example, the unique local IPv4 address 192.168.1.1 is generated as the IP address.

3.1.2 Protocols
The OPC UA bus controller uses 2 types of protocols for communication:

Binary protocol (UA-TCP UA-SC UA-Binary)
Advantages of the binary protocol:
  • Very good performance
  • Low resource consumption
  • Uses only one port (4840) for communication

Web server log
This protocol is only used in the bus controller for firmware downloads via the web server.

Structure of the protocols
### 3.1.3 Establishing a connection with UA_Connect() via hostname

By default, it is not possible to establish a connection to the hostname using function block `UA_Connect()`. The following additional settings on the CPU are necessary for this:

![DNS parameters](image)

**Hostname example**

- **ST program**
  
  ```
  ServerEndpointUrl := 'opc.tcp://bropc229:4840';
  ```

- **UaExpert**
  
  `opc.tcp://bropc229:4840`

### 3.2 Object access and information model

All bus controllers and module objects (including directories, data, methods, I/O data, etc.) can be referenced uniquely in the following ways:

- Via the directory path
- Object name
- NodeID

The NodeID is a basic requirement for accessing object data or methods using library "AsOpcUac" on the OPC UA bus controller. These NodeIDs can be determined from directory paths and object names using function block `UA_TranslatePath()`.

![UA_TranslatePath](image)

"Root" or "Objects" objects are suitable as StartNodeID for the function block since they have fixed NodeIDs and form the top level of the information model.

- NodeID (Root) = 84
- NodeID (Objects) = 85
- NamespaceIndex = 0
- IdentifierType = Numeric

**Example of a NodeID entry**

To shorten relative path names, the StartNodeID can be moved to subobjects instead of "Root" or "Objects".
3.2.1 NodeID, BrowseName and NamespaceIndex

The OPC UA bus controller contains the following types of nodes:

- Nodes that are prescribed according to the specifications (for example, server object).
- System-specific nodes created during startup of the OPC UA server (e.g. tasks and variables of tasks).
- All type descriptions of the system-specific nodes are also mapped as nodes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeID</td>
<td>UANodeID</td>
<td>A node is uniquely identified in the address space via its NodeID. The NodeID consists of a NamespaceIndex, IdentifierType and Identifier.</td>
</tr>
<tr>
<td>NamespaceIndex</td>
<td>UINT</td>
<td>Controls the responsibility of the node. See “Namespaces used” on page 18.</td>
</tr>
<tr>
<td>IdentifierType</td>
<td>UINT</td>
<td>Specifies whether the identifier is represented by a numeric value or defined by a character string.</td>
</tr>
<tr>
<td>Identifier</td>
<td>STRING[255]</td>
<td>Can be displayed in the following way:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Numerical value, e.g. 2253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- String, e.g. “SubDevices”</td>
</tr>
</tbody>
</table>

**Information:**
Numerical values are also entered as strings in the bus controller. (e.g. “256”)

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrowseName</td>
<td>Qualified name</td>
<td>If it is not known which nodes a server contains, then the address space can be searched starting from the root node or another known node. When searching, the node found is identified by the BrowseName.</td>
</tr>
</tbody>
</table>

**Information:**
The NamespaceIndex of the BrowseName can be in a different namespace than NamespaceIndex of the associated NodeID.

<table>
<thead>
<tr>
<th>Name</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces</td>
<td></td>
<td>Since predefined nodes of different organizations or companies can have identical identifiers, a namespace is also specified. This allows identical node identifiers to be uniquely assigned. The namespace specifies which organization or company defined the node and is specified in the form of a namespace URI. For more information about predefined namespaces, see “Namespaces used” on page 18.</td>
</tr>
</tbody>
</table>

For most nodes of the OPC UA bus controller, the NamespaceIndex of the "BrowseName" and "NodeID" is identical. Exceptions are possible, for example, in the attributes of the parameters of bus controller methods. When browsing in a system, these exceptions are automatically taken into account since the corresponding NamespaceIndex is also returned for the "BrowseName".

**Information:**
By default, the NamespaceIndex determined using UA_GetNamespaceIndex(), for example, can also be specified for the "BrowseName". However, the corresponding NamespaceIndex must be determined separately for the parameters of the bus controller methods.

**Examples**

**NodeID**

NodeID consisting of NamespaceIndex, IdentifierType and Identifier

```
NodeID
  NamespaceIndex 2
  IdentifierType Numeric
  Identifier 11
```

**BrowseName**

BrowseName with associated NamespaceIndex

```
NodeID
  NamespaceIndex 2
  IdentifierType Numeric
  Identifier 11
```

```
BrowseName 2 "SubDevices"
```

* The NamespaceIndex is placed before the BrowseName, separated by a comma. If it differs from the NamespaceIndex shown above, this must be taken into account in the RelativePath.

** The NamespaceIndex refers to the NodeID to be determined.
3.2.2 Namespaces used

The following namespaces are used in the OPC UA bus controller:

<table>
<thead>
<tr>
<th>Namespace URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://opcfoundation.org/UA/">http://opcfoundation.org/UA/</a></td>
<td>NamespaceIndex 0(^1) Types and objects specified by the OPC Foundation</td>
</tr>
<tr>
<td>urn:br-automation/BuR/UA/X20BC008U</td>
<td>B&amp;R NamespaceIndex 1(^1)</td>
</tr>
<tr>
<td><a href="http://br-automation.com/OpcUa/BC/io-system/">http://br-automation.com/OpcUa/BC/io-system/</a></td>
<td>B&amp;R NamespaceIndex 2(^2) Information model for the bus controller I/O system</td>
</tr>
</tbody>
</table>

1) Standardized, NamespaceIndex cannot be changed.
2) This NamespaceIndex is not standardized and can therefore be changed. This means that the currently used index "2" could change in future firmware versions if necessary.

To avoid possible problems in applications, the namespace used should always be read with UA_GetNamespaceIndex() and added to the relative path name as a variable string (currently "2") before the directory or object name.

The namespaces used can also be read from the bus controller.
3.2.3 Difference BrowseName and DisplayName

For most nodes, the BrowseName and DisplayName are identical. However, a changed DisplayName can be used to provide better information.

If the OPC UA bus controller browses the information model, the DisplayName is displayed although the BrowseName must be specified for the actual browse process.

3.2.3.1 Different BrowseNames in directory "X2X / SubDevices"

Directory "X2X / SubDevices" displays the connected or configured X2X modules.

To achieve a clear and meaningful display, leading zeros are added in front of the slot in the DisplayName and the module designation is also displayed. However, this name is not suitable for a browse process since it is too complex and requires knowledge of the module data.

* The NamespacelIndex is placed before the BrowseName, separated by a comma. If it differs from the NamespacelIndex shown above, this must be taken into account in the RelativePath.

** The NamespacelIndex refers to the NodeID to be determined.

For directory "SubDevices", however, there is a unified BrowseName that has a very simple structure and does not require any previous knowledge of the module data. This applies to all X2X slots:

- Relative path + "/NamespacelIndex:ST" followed by the slot without leading zeros.

Examples

/2:ST1 For slot 1
/2:ST23 For slot 23

Programming example for slot 3

StartNodeID set to UA_StartPath '/0:Root/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices';
NamespaceIndex_BC_Str := '2';          // define via UA_GetNamespaceIndex()

// generate RelativePath to Object at slot# 3
IO_Slot := 3;                          // Module slot number
itoa(IO_Slot,ADR(IO_Slot_String));     // Prepare slot string

// generate RelativePath
// Path := '/2:ST3';
RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ST'));
strcat(ADR(RelativePath_ObjectNode),ADR(IO_Search_Slot)); // add prepared search slot string
3.2.3.2 Object access via "Configuration / General / DigitalOutputsPacked"

In this example, the data format (packed/unpacked) of the digital outputs of module "X20DO9322" is accessed in slot 34 "ST034" via directory "Configuration".

Directories for node accesses

Node access is usually possible via different directories or object overviews. In this example, it is directory "Configuration". Object overview "ParameterSet" is best suited, however, since all module-specific data points are available here.

Determining the information for object access

- **DisplayName** "Packed outputs" is displayed (framed in blue). This is different from the **BrowseName** required to specify the path, however.
- The **"BrowseName"** (framed in green) consists of 2 components: the **NamespaceIndex** (in this example "2") and the **BrowseName** (in this example "DigitalOutputsPacked").
- Both the **BrowseName** and **DisplayName** can contain spaces in the name.
- You can display or change the value of node "Value" node (framed in red). This node is of data type "Enumeration" and requires DINT storage space.
- The exact data type of the enumeration is "X20DOx322_OnOff" (framed in yellow).

![Diagram](image)

The **NamespaceIndex** is placed before the **BrowseName**, separated by a comma. If it differs from the **NamespaceIndex** shown above, this must be taken into account in the **RelativePath**.

Programming example

```
StartNodeID set to UA_StartPath '/0:Root/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices';
NamespaceIndex_BC_Str := '2';          // define via UA_GetNamespaceIndex()
// generate RelativePath to Object at Slot# 3
IO_Slot := 34;                           // Module slot number
itoa(IO_Slot,ADR(IO_Slot_String));      // Prepare slot string
// generate RelativePath
// Path := '/2:ST34';
RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ST34'));                         // Module slot number
strcat(ADR(RelativePath_ObjectNode),ADR(':Configuration/'));
strcat(ADR(RelativePath_ObjectNode),ADR(':General/'));
strcat(ADR(RelativePath_ObjectNode),ADR(':DigitalOutputsPacked'));
```
3.2.4 NamespaceIndex 0

The following list shows all exceptions to the OPC UA bus controller default namespace 2 "http://br-automation.com/OpcUa/BC/io-system". All these object nodes use NamespaceIndex 0 according to the standard.

- Root (for StartNodeId)
- Objects (for StartNodeId)
- Folder "Types" with all associated nodes
- Folder "Views" with all associated nodes
- OutputArguments
- InputArguments

**Information:**

When browsing in OPC UA systems, the required NamespaceIndex is returned to the "BrowseName"; these exceptions are therefore taken into account automatically.

If the BrowseName "InputArguments" or "OutputArguments" of methods is required, NamespaceIndex 0 must be referenced.

* Exception: NamespaceIndex 0 for BrowseName of the input and output arguments.

**Determining the NamespaceIndex**

- In directory "Objects", the NamespaceIndex determined by UA_GetNamespaceIndex(), for example, can be specified by default for the BrowseName.
- In directories "Types" and "Views", the NamespaceIndex can come from different namespaces and must therefore be determined manually using "Browse".
- With the parameters of the bus controller methods, the associated NamespaceIndex must be determined separately.
### 3.2.5 StartNodeID

By default, all NodeIDs and names from NamespaceIndex 0 remain unchanged. These are therefore suitable for use as start index "StartNodeID".

The node to be determined is defined via the StartNodeID and RelativePath.

**Important StartNodeID.Identifier from NamespaceIndex 0**

<table>
<thead>
<tr>
<th>StartNodeID.Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'84'</td>
<td>&quot;Root&quot; from NamespaceIndex 0</td>
</tr>
<tr>
<td>'85'</td>
<td>&quot;Objects&quot; from NamespaceIndex 0</td>
</tr>
</tbody>
</table>

**Display of StartNodeID directory "Root" in UaExpert**

BrowseName and DisplayName are identical.

```c
(* Automation Studio Struct Type: UANodeID *)
(* set StartNodeID to <Root> *)
StartNodeID.NamespaceIndex := 0;
StartNodeID.Identifier := '84'; // <Root>
StartNodeID.IdentifierType := UAIdentifierType_Numeric;
```

**Display of StartNodeID directory "Root / Objects" in UaExpert**

BrowseName and DisplayName are identical.

```c
(* Automation Studio Struct Type: UANodeID *)
(* set StartNodeID to <Root><Objects> *)
StartNodeID.NamespaceIndex := 0;
StartNodeID.Identifier := '85'; // <Objects>
StartNodeID.IdentifierType := UAIdentifierType_Numeric;
```
3.2.6 Relative path names

The relative path name "RelativePath" is required when using the function blocks of OPC UA client library "AsOpcUac" to determine nodes. See "UA_TranslatePath()" on page 76. Nodes are always defined via the StartNodeID and the relative path. The relative path name refers to a StartNodeId (framed in green) from which the relative path begins. To determine a NodeId via the relative path name or an application-specific start index, a StartNodeID specified by the standard can be used. See "StartNodeID" on page 22.

For information about the data types of node attribute "BrowseName" or "DisplayName", see "Detailed information about DataType "QualifiedName" and "LocalizedText"" on page 28.

Using NamespaceIndex values in relative path name "2"

The use of the permanently coded NamespaceIndex 2 should be avoided since it may change in later bus controller firmware versions. To avoid possible problems with later firmware versions, the used namespace should be read out with UA_GetNamespaceIndex() and added to the relative path name as a variable string, e.g. "2" before the directory or object name.

The specification of relative path names is standardized according to the BNF format.

Information:

If the NamespaceIndex differs from the namespace determined with UA_GetNamespaceIndex(), it is listed in section "NamespaceIndex 0" on page 21.

Examples

Use of permanently coded path names

The following simplified definition of the relative path to "DeviceSet / X20BC008U / X2X / SubDevices" should not be used.

RelativePath_ObjectNode:= '/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices';

Using a relative path name

The following definition of the relative path to "DeviceSet / X20BC008U / X2X / SubDevices" can be easily adjusted in the event of a namespace change.

NamespaceIndex_BC_Str := '2'; // String contains function block data from 'UA_GetNamespaceIndex'

RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR{'DeviceSet/'});
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR{'X20BC008U/'});
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR{'X2X/'});
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR{'SubDevices'});
Programming tip

If the string definitions of a directory or object name are defined globally only once in an application, it only has to be adjusted in one place when the name is changed.

In addition, an extension of the NamespacelIndex string from "2" to "/2:" optimizes further definition and programming work.

```plaintext
// Path := '/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices';
NamespaceIndex_String := '/2:';    // String contains function block data from 'UA_GetNamespaceIndex'
RelativePath_ObjectNode := NamespaceIndex_String;
strcat(ADR(RelativePath_ObjectNode),ADR(UA_REL_PATH_DEVICESET));       // 'DeviceSet'
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_String));       // X20BC008U'
strcat(ADR(RelativePath_ObjectNode),ADR(UA_REL_PATH_X2X));             // 'X2X'
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_String));       // SubDevices'
```

### 3.2.6.1 Long path names

In this example, an interface parameter of module X20CS1020 in slot 44 "ST044" should be accessed via directory "Configuration / Transmit frame configuration".

The resulting relative path name for this example is: "/0:Root/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST44/2:Configuration/2:TransmitConfiguration/2:TxFrameLengthMax".

These types of long path names can be shortened using a favorable StartNodeIndex selection. The recommended StartNodelndex is 1 or 2 levels above the actual node.

```plaintext
// StartNodeID := '/0:Root/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST44';
// Path := '/2:Configuration/2:TransmitConfiguration/2:TxFrameLengthMax';
```

### Directories for node accesses

Node access is usually possible via different directories or object overviews. In this example, it is directory "Configuration". Object overview "ParameterSet" is best suited, however, since all module-specific data points are available here.

### Determining the information for object access

- The display names displayed are different from the BrowseNames required to specify the path.
  - DisplayName "Maximum length of frame" (framed in blue)
- The "BrowseName" (framed in green/yellow) consists of 2 components: the NamespacelIndex and BrowseName.
  - NamespacelIndex "2"
  - BrowseName "TxFrameLengthMax"
- Both the BrowseName and DisplayName can contain spaces in the name.
- The NamespacelIndex marked red in "NodeID" may differ from the NamespacelIndex of the browse name (marked yellow/green) (see "Determining DataType for method calls" on page 27).
- Attributes "Value" and "DataType" (bottom 2 lines) of the node can be read out.
Basic information

Programming example

If necessary, a different NamespaceIndex is possible before each new browse name. The different memory locations for the NamespaceIndex allow referencing to BrowseNames with different NamespaceIndexes. In this example, however, all browse names are from NamespaceIndex 2.

StartNodeID set to UA_StartPath '/0:Root/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST44';
NamespaceIndex_BC_Str := '2'; // define via UA_GetNamespaceIndex()
NamespaceIndex_Target1_Str := '2'; // would be correctly defined via Browse
NamespaceIndex_Target2_Str := '2'; // would be correctly defined via Browse

// generate RelativePath
// Path := '/2:Configuration/2:TransmitConfiguration/2:TxFrameLengthMax';
RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':Configuration/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_Target1_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':TransmitConfiguration/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_Target2_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':TxFrameLengthMax'));

3.2.6.2 Different NamespaceIndex

There is only a different NamespaceIndex for nodes directly from the root directory or for the BrowseNames of method arguments (see "Determining DataType for method calls" on page 27). All exceptions to NamespaceIndex 2 for the OPC UA bus controller are listed in NamespaceIndex 0.

For an example, see "UA_GetNamespaceIndex()" on page 75.
3.2.7 Data Type

Data type "DataType" of the selected node on the OPC UA client (e.g. Automation Studio program) must match the node data type on the OPC UA server (bus controller). Otherwise, an error message with the corresponding error number is displayed during access. This applies to both read and write accesses.

Overview of all data types

<table>
<thead>
<tr>
<th>Designation on OPC UA bus controller</th>
<th>Value</th>
<th>Designation in Automation Studio</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VariantType</td>
<td></td>
<td>UAVariantType</td>
<td>Defines a variant data type and the structure that contains the variant data.</td>
</tr>
<tr>
<td>Boolean</td>
<td>1</td>
<td>BOOL</td>
<td>Boolean data type</td>
</tr>
<tr>
<td>SByte</td>
<td>2</td>
<td>SINT</td>
<td>1-byte signed integer</td>
</tr>
<tr>
<td>Byte</td>
<td>3</td>
<td>USINT</td>
<td>1-byte unsigned integer</td>
</tr>
<tr>
<td>Int16</td>
<td>4</td>
<td>INT</td>
<td>2-byte signed integer</td>
</tr>
<tr>
<td>UInt16</td>
<td>5</td>
<td>UINT</td>
<td>2-byte unsigned integer</td>
</tr>
<tr>
<td>Int32</td>
<td>6</td>
<td>DINT</td>
<td>4-byte signed integer</td>
</tr>
<tr>
<td>UInt32</td>
<td>7</td>
<td>UDINT</td>
<td>4-byte unsigned integer</td>
</tr>
<tr>
<td>Float</td>
<td>10</td>
<td>REAL</td>
<td>Floating-point number (4 bytes)</td>
</tr>
<tr>
<td>Double</td>
<td>11</td>
<td>LREAL</td>
<td>Double</td>
</tr>
<tr>
<td>String</td>
<td>12</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>DateTime</td>
<td>13</td>
<td>DATE_AND_TIME</td>
<td>Time and date</td>
</tr>
<tr>
<td>ByteString</td>
<td>15</td>
<td>UABYTESTRING</td>
<td>Structure for data definition (e.g. for X20CS modules)</td>
</tr>
<tr>
<td>NodeID</td>
<td>17</td>
<td>UANODEID</td>
<td>NodeID of the node, identification of the object</td>
</tr>
<tr>
<td>ExpandedNodeID</td>
<td>18</td>
<td>UAEXPLETEDNODEID</td>
<td>Structure for defining an ExpandedNodeID</td>
</tr>
<tr>
<td>QualifiedName</td>
<td>20</td>
<td>UAQUALIFIEDNAME</td>
<td>Structure for defining a QualifiedName</td>
</tr>
<tr>
<td>LocalizedText</td>
<td>21</td>
<td>UALOCALIZEDTEXT</td>
<td>Structure for defining a LocalizedText</td>
</tr>
<tr>
<td>DataValue</td>
<td>23</td>
<td>UADATAVALUE</td>
<td>Structure for defining DataValue</td>
</tr>
<tr>
<td>ExtensionObject [Argument]</td>
<td>296</td>
<td>UAMETHODARGUMENT</td>
<td>Input and output arguments of methods</td>
</tr>
</tbody>
</table>

Data buffer type

ByteString → UAByteString (Automation Studio)

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>DINT</td>
<td>Current length of data</td>
</tr>
<tr>
<td>Data</td>
<td>USINT[0 .. MaxIndex_Bytestring]</td>
<td>Data buffer (default maximum 1024 bytes)</td>
</tr>
</tbody>
</table>

Input and output arguments of methods

ExtensionObject [Argument] → UAMethodArgument (Automation Studio)

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING[64]</td>
<td>Name of the method argument</td>
</tr>
<tr>
<td>Value</td>
<td>STRING[255]</td>
<td>Value of the method argument</td>
</tr>
</tbody>
</table>

Examples
3.2.7.1 Determining DataType for method calls

To call methods using library "AsOPCUAc", the data types of the individual input and output arguments as well as the associated data types must be known in Automation Studio.

UaExpert makes it easy to determine both the number of arguments required and their data type.

- **Determining the number of arguments** (1)

  The number of input or output arguments required can be read for the respective argument under "Attribute / Value /" → Value ArgumentArray[x]. "[x]" stands for the number of arguments necessary for each method call.

- **Determining the data type of the array argument** (2 and 3)

  UaExpert displays a separate sub-entry under "Attribute / Value / Value" for each necessary argument. This consists only of the zero-based index number, e.g. "[0]". Clicking on the entry displays information about the array argument. To get information about the data type itself, "DataType" must be opened by clicking on it.

- **Determining the data type for Automation Studio**

  The UaExpert data type must be adjusted to define the data type in Automation Studio. This is possible using the number in front of the data type identifier.

  **Example**

<table>
<thead>
<tr>
<th>Name</th>
<th>&quot;DataType&quot; &quot;Identifier&quot;</th>
<th>Automation Studio &quot;Type&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>15 [ByteString]</td>
<td>UAByteString</td>
</tr>
<tr>
<td>Bytes written</td>
<td>6 [Int32]</td>
<td>DINT</td>
</tr>
</tbody>
</table>

  To assign the DataType names used in OPC UA to the data types available in Automation Studio, see "DataType" on page 26.
• Variable declaration in Automation Studio

Once the data type for Automation Studio has been determined, it can be declared in Automation Studio.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOInputArguments_7</td>
<td>UAInputArgument[0-9]</td>
</tr>
<tr>
<td>iOOutputArguments_7</td>
<td>UAOuputArgument[0-9]</td>
</tr>
<tr>
<td>CS1020_WriteBufferLen_7</td>
<td>UInt</td>
</tr>
<tr>
<td>CS1020_WriteByteCnt_7</td>
<td>UInt</td>
</tr>
</tbody>
</table>

3.2.7.2 Detailed information about DataType "QualifiedName" and "LocalizedText"

When reading an attribute using the AttributeID of a node (e.g. AttributeID = UAAI_BrowseName or AttributeID = UAAI_DisplayName), the correct data type of the target variable must be specified. These data types are not displayed in UaExpert.

**Information:**

The server and client data types must be identical!

**BrowseName**

When reading "BrowseName" (AttributeID = UAAI_BrowseName) from any node, data type "QualifiedName" → "UAQualifiedName" (Automation Studio) must be defined as the target variable for the client.

**DisplayName**

When "DisplayName" (AttributeID = UAAI_DisplayName) of any node is read out, data type "LocalizedText" → "UALocalizedText" (Automation Studio) must be defined as the target variable for the client.

**Model data type**

When "Value" (AttributeID = UAAI_Value) of node "Model" is read, data type "LocalizedText" is required. In this case, the necessary data type is displayed below Value.
3.2.7.3 Detailed information about data type Enumeration

The enumeration format "enum" is always mapped as a 4-byte value (DINT). It can therefore also be accessed without knowledge of the exact enumeration type.

All enumeration definitions can be read from the following directory on the bus controller: "Root / Types / DataTypes / BaseDataType / Enumeration"

Example

Attribute value "Value" of node "EnumStrings" (NodeID: 2 / 61457) must be accessed with process variable data type "LocalizedText Array[2]" or greater.

3.2.7.4 Details about DataType ExtensionObject "UAMethodArgument"

Input and output arguments of methods

ExtensionObject [Argument] → UAUAMethodArgument (Automation Studio)

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>STRING[64]</td>
<td>Name of the method argument</td>
</tr>
<tr>
<td>Value</td>
<td>STRING[255]</td>
<td>Value of the method argument</td>
</tr>
</tbody>
</table>

The OPC UA Bus Controller argument name is specified in structure element "Name". Element "Value" references the process variable in the client program and specifies the name of the variable in node identification format.

For more information, see "Executing methods using library "AsOpcUac"" on page 54.

3.2.7.5 Notes about digital I/Os in PACKED format

When reading and writing digital I/Os in PACKED format, the following must be observed to ensure error-free and proper access to a memory location:

- Up to bus controller firmware version 1.06:
  All data is of data type UINT (2 bytes).
- Bus controller firmware version 1.06 and later:
  All data less than or equal to 8 channels is of data type USINT (1 byte).
  All data with more than 8 channels is of data type UINT (2 bytes).
3.2.8 Access attributes

Function block `UA_Read()` can be used to access not only the values of a `NodeID`, but also the attributes selected in `NodeAddInfo`. All possible attributes are listed below.

Depending on the data type, only a subset of the listed attributes is available for each `NodeID`. Unsupported attributes are rejected during read access with a corresponding error message.

<table>
<thead>
<tr>
<th>AttributeID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAAL_Value</td>
<td>The value of a variable.</td>
</tr>
<tr>
<td>UAAL_DataType</td>
<td>Node ID for the data type of the variable value.</td>
</tr>
<tr>
<td>UAAL_ValueRank</td>
<td>The number of dimensions in the value.</td>
</tr>
<tr>
<td>UAAL_ArrayDimensions</td>
<td>The length for each dimension of an array value.</td>
</tr>
</tbody>
</table>

**Example**

The value of a `NodeID` can be read with AttributeID "UAAL_Value". AttributeID ID "UAAL_ArrayDimensions" is used to read out the array length (e.g. of the bus controller status array).
3.2.9 Node identification

The address space of the information models for process variables is dynamic and determined by the application modules, tasks and variables in the application. The node identification is formed from the names of these objects. If the name or scope of an object does not change, this identification remains unchanged after a restart.

The following table describes the syntax of addressing for the possible objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Addressing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application module</td>
<td>An application module is addressed by its name followed by &quot;:&quot;. The default application module is an empty string.</td>
</tr>
<tr>
<td>Task</td>
<td>A task is addressed using the name of the application module and the name of the task.</td>
</tr>
<tr>
<td>Process variable</td>
<td>A process variable is addressed using the name of the task followed by &quot;::&quot; and the name of the variable. For global variables, the name of the global variable is specified after the application module. A process variable is addressed in the same way for scalars, arrays and structures.</td>
</tr>
<tr>
<td>Process variable array element</td>
<td>In order to address an element of an array, its index must be specified. For multidimensional arrays, the indexes of the desired element are separated by commas. The indices are within the limits specified in the array declaration. For arrays with negative dimensions, it is possible to specify an appropriate negative value.</td>
</tr>
<tr>
<td>Process variable structure element</td>
<td>To address elements of a structure, a path is derived from the names of the nested elements. The individual elements are separated by a period. For arrays with structures, an element can be selected with &quot;[]&quot;.</td>
</tr>
</tbody>
</table>

Process variables are used as function module transfer variables and must therefore be declared in advance with the correct data type.

In order to make process variables accessible as data points for library AsOpcUac, they must be enabled as OPC UA tags first. Addressing is different for local and global variables.

**Local variables**
Local process variables are addressed with ":::TaskName:VariableName".

**Global variables**
Global process variables are addressed with ::VariableName.

**Example**

<table>
<thead>
<tr>
<th>ObjectName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WidgetTests</td>
<td>Data types</td>
</tr>
<tr>
<td>UAc_ConfigurationStatusAll</td>
<td>Description</td>
</tr>
</tbody>
</table>

Function block in/out variable: :::Program:localVar

Function block in/out variable: ::globalVar

**Examples**

```
(* INITIAL SETUP before UA_Read() *)
Variable_Rd0 := ':::OpcTest:BC_SoftwareRevision';
BC_SoftwareRevision := 'V?.??'; // init Target Variable for UA_Read()

(* INITIAL SETUP before UA_ReadList() *)
Variable_RdList_0[0] := ':::OpcTest:UAc_ConfigurationStatusAll';
UAc_ConfigurationStatusAll := 0; // init Target Variable for UA_ReadList()
```
Information:
For read or write access, a process variable that corresponds to the data type must be declared in the Automation Studio program and used at least once in the program (e.g. initializing the variable in the initialization subroutine).

3.2.10 Monitored item

Monitored items are monitored by the slave and sent to the client for updating at regular intervals or after a change in value.

Display in UaExpert

Each entry in the list is a monitored item. In addition, the parameter values can be displayed in a separate window.

Information:
With bus controller firmware version 1.06 and later, a maximum of 1000 monitored items are supported.

3.2.11 Subscription

A subscription contains a list of monitored items that are updated at regular intervals or after value changes. All value changes that have occurred are transmitted collectively after the minimum update time at the latest. The total update of a subscription can be switched on or off separately.

Display in UaExpert

Display of a subscription. In addition, a parameter window can be opened.

Information:
With bus controller firmware version 1.06 and later, a total of 10 subscriptions can be handled. Only a maximum of 5 subscriptions can be registered for a connection, however.
3.2.12 Object access via handle

As soon as the NodeId needed for an access is known, a handle can be reserved for it that is retained for the entirety of the session. Depending on the type of the selected node, this handle can be used for read, write or execute access.

3.2.13 Automated access to the information model

The actual access to the nodes in the information model always takes place via the NodeID. This NodeID consists of the numeric 16-bit NamespaceIndex, the IdentifierType and 32-bit Identifier according to the OPC UA standard. OPC UA bus controller X20BC008U supports only IdentifierType "Numeric".

When accessing nodes using a software library (e.g. AsOpcUac), the following must be observed:

- First, the namespace strings should be translated into the NamespaceIndex. For this purpose, client libraries provide corresponding functions, e.g. function block UA_GetNamespaceIndex().
- Node IDs should be determined using function block UA_TranslatePath().

**Information:**

Using hard-coded node IDs can cause problems since they may change with a later firmware version.

- A handle must be reserved before the actual access. This handle is retained throughout the entire session until freed up.
- The numerical value of the identifier must be passed in "String" format, e.g. identifier 12 is specified as string "12".
- Access via lists or subscriptions is more efficient than single accesses. For digital inputs and outputs, compressed data format "Packed" can be configured.
4 Bus controller objects and directories

The following figures provide an overview of the objects and directories available in the OPC UA bus controller.

Legend
The following icons are used in the object dictionary:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍃</td>
<td>Directory, folder</td>
</tr>
<tr>
<td>🍃🍃</td>
<td>Object</td>
</tr>
<tr>
<td>🍃🍃🍃</td>
<td>Variable or parameter</td>
</tr>
<tr>
<td>🍃🍃🍃🍃</td>
<td>Property variable (additional information)</td>
</tr>
<tr>
<td>🍃🍃🍃🍃🍃</td>
<td>Method</td>
</tr>
</tbody>
</table>

4.1 Configuration

Both the bus controller and each connected or configured X2X module have directory "Configuration". The following subdirectories and subobjects also exist:

- **Control**  Lists the methods available for configuration
- **Network**  Contains information about the network configuration
- **Timer**    Contains timer and timestamp information
4.1.1 Control

This directory contains the methods available for configuring the bus controller or X2X module. For an overview of methods, see "Methods" on page 52.

Information:
Directory MethodSet should be used to call methods.

4.1.2 Network

A valid network configuration is required in order for the bus controller to operate. The network configuration used is based on the address switch setting and the configuration stored in flash memory. For more information, see "Overview of network address switch functionality" on page 7.

For new configuration data to be transferred, it must be saved in the control object by calling method Apply in the control object.

Information:
The network parameters will only be applied after the bus controller is restarted.

Position of the data in the information model: "/X20BC008U/Configuration/Network"

<table>
<thead>
<tr>
<th>Node name</th>
<th>Description</th>
</tr>
</thead>
</table>
| DHCP                       | Enables/Disables DHCP client functionality
If the DHCP client is enabled, the gateway, IP address, subnet mask and DNS server parameters are obtained from the DHCP server. In this case, the configured values are not used. They can be read and modified, however. |
| Gateway                    | Configuration of the default gateway IP address                                                 |
| Hostname                   | Configuration of the hostname                                                                  |
| IP address                 | Configuration of a static IP address                                                           |
| Primary DNS / Secondary DNS| Configuration of a primary or secondary DNS server                                               |
| Subnet mask                | Sets the subnet mask                                                                            |
4.1.3 Timer

OPC UA uses the UTC timestamp. It is possible to configure multiple NTP servers for use as time references.

**Information:**

Both NTP and the OPC UA server transfer the UTC timestamp. If UTC time needs to be converted to a corresponding local time, the conversion must be carried out via the OPC UA client.

Position of the data in the information model: "../ X20BC008U / Configuration / Timer"

<table>
<thead>
<tr>
<th>Node name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableNTP</td>
<td>Enables/Disables time synchronization.</td>
</tr>
<tr>
<td>JumpDifference</td>
<td>Time difference from which a time jump occurs (in milliseconds). During normal operation, the internal timer synchronizes to the reference time of at least one time server. If a difference is detected that is greater than the value specified in this parameter, a time jump occurs. If the deviation is minimal, a time jump does not occur. Instead, the difference is corrected by adjusting the timer speed.</td>
</tr>
<tr>
<td>SynchronizationInterval</td>
<td>Interval at which the time is polled from the time server (in minutes).</td>
</tr>
<tr>
<td>TimerServer01/TimerServer02</td>
<td>Hostnames or IP addresses of up to 2 time servers. If one of the servers is not available, the other will be used instead.</td>
</tr>
<tr>
<td>TryDNS</td>
<td>An attempt will be made to use the DNS server as the time server. This option can be useful if the network configuration is obtained via DHCP since network stations on which the DNS server is hosted will often host an NTP server as well. In this case, an NTP server does not need to be configured.</td>
</tr>
<tr>
<td>TryGateway</td>
<td>An attempt will be made to use the default gateway as the time server. This option can be useful if the network configuration is obtained via DHCP since the default gateway often hosts an NTP server as well. In this case, an NTP server does not need to be configured.</td>
</tr>
</tbody>
</table>
4.2 Identification

In node X20BC008U as well as in X2X modules, general additional information specific to the OPC UA standard is provided as property variables.

Object X20BC008U and the objects that represent the I/O modules are subordinate to several nodes that provide information about the respective module.

<table>
<thead>
<tr>
<th>Node name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceManual</td>
<td>URL to additional information about the module</td>
</tr>
<tr>
<td>DeviceRevision</td>
<td>Reserved</td>
</tr>
<tr>
<td>HardwareRevision</td>
<td>Hardware revision of the module</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Manufacturer of the module</td>
</tr>
<tr>
<td>Model</td>
<td>Product ID</td>
</tr>
<tr>
<td>RevisionCounter</td>
<td>Reserved (always -1)</td>
</tr>
<tr>
<td>Serialnumber</td>
<td>Full serial number as a string</td>
</tr>
<tr>
<td>HardwareRevision</td>
<td>Softwarerevision</td>
</tr>
<tr>
<td>Identification / Module ID</td>
<td>Numeric identification number for the module</td>
</tr>
<tr>
<td>Identification / Vendor ID</td>
<td>Numeric ID of the module manufacturer</td>
</tr>
</tbody>
</table>
4.3 MethodSet and ParameterSet

The bus controller object and the I/O module objects each contain objects "MethodSet" and "ParameterSet".

- "MethodSet" groups all methods for a given module into a flat list.
- "ParameterSet" groups all variable nodes into a flat list.

These sets should be used if variables and nodes are to be accessed automatically. The advantage of providing access via a set is that the location of the nodes is known from the start. It is then not necessary to take an additional structure into account. This allows the structure to be changed (e.g. due to new features, better clarity, etc.) without having to adapt automatic accesses.

Position of the data in the information model:

"../ X20BC008U / MethodSet"
"../ X20BC008U / ParameterSet"

For an overview of methods, see "Methods" on page 52.
4.4 Process data of the bus controller

This node can be used to read the embedded parameter chip of OPC UA bus controller X20BC008U. It contains basic module information such as serial number and firmware version. This information is grouped in folder "ProcessData".

**Information:**

This function is only available with firmware version V1.18 or later.

In older firmware versions, this function was only available for X2X modules (see "Process data of the I/O modules" on page 42).

**Process data of OPC UA bus controller X20BC008U**

- **Serial number**

  The node named "SerialNumber" occurs twice in directory X20BC008U with different meanings.
  - (1) In directory "ProcessData".
    This node is of data type UDINT (UInt32) and contains the serial number as a decimal value (e.g. 168481).
  - (2) In directory "X20BC008U".
    This node is required by the standard, is of data type STRING and contains the serial number as a string (e.g. "EAFB0168481"). This number is identical to the serial number printed on the outside of the module.

**4.5 Status**

This directory contains current status information about the bus controller or I/O module. For more information, see "Configuration status" on page 46.
4.6 X2X Link and I/O modules

All I/O modules detected on the X2X Link network are listed under "X20BC008U / X2X / SubDevices". All data, control and configuration nodes assigned to the respective module are located at a lower level.

Example of an AI module's nodes

Connected modules are automatically detected and started with their default configuration, provided that a corresponding configuration has not been saved. In order to fully map I/O module nodes, the bus controller contains a database with additional information for each module. The information model is updated accordingly, allowing the modules' full range of functions to be used. For a list of modules supported by the database, see "Supported modules" on page 12.

X2X modules not contained in the database can be used in function model 254. Since the bus controller does not know the product ID in this case, the modules are labeled "ID 'ModuleID>'" (see e.g. "ID 41870" in the image above). The configuration for these modules can only be modified using method "RegisterWrite" on page 61. The process data points are labeled "RegisterXX" in folder Process data of the I/O modules. For the meaning of register numbers, see the relevant module documentation.

**Information:**

It is not possible to modify the process data image in this operating mode.
4.6.1 ConfigChannels

This folder groups nodes containing a copy of the values that were written to the relevant configuration register in the module. These values can only be read and are provided for troubleshooting purposes. They are automatically calculated on initialization or on applying the configuration. For more information about the meaning of these values, see the relevant module documentation.

Example of an AI module's ConfigChannel nodes

---

4.6.2 Configuration

Not only the bus controller but also each individual I/O module can be configured via the information model. To enable this, a separate configuration folder is included for each I/O module.

Method directory "Control" is available for controlling the configuration. This directory contains the methods available for configuring the X2X module. For an overview of methods, see "Methods" on page 52.

The other configuration parameters contained in this folder and their structure are individual for each I/O module. For an explanation of parameters, see the respective module documentation.

For querying the current configuration status, see "Evaluating the configuration status" on page 47.

4.6.3 Diagnostics

The data from this subdiractory is used for diagnostic purposes, such as internal system analyses.

4.6.4 Identification

For information about this directory, see "Identification" on page 37.
4.6.5 Process data of the I/O modules

The process data provided by an I/O module is grouped in folder "ProcessData". These nodes allow output process data to be written and input process data to be read. For the meaning of data points, see the relevant module documentation.

Example of an AI module's process data

![Diagram of ProcessData folder with subfolders]

**Information:**

If the configuration is modified, the available process data nodes may change. To detect this, the relevant higher-level node must be re-imported (browsed).

Until the relevant configuration changes are applied, it may not be possible to feed data to recently enabled nodes. This is indicated via a corresponding state.

4.6.6 Status

For information about this directory, see "Status" on page 39.
5 Bus controller configuration

The OPC UA bus controller configuration includes several configuration types.

- The network configuration depends largely on the position of the address switch. See "Network address switches" on page 7.
- All I/O modules connected on the X2X Link network are entered in the SubDevices list and automatically initialized and started by the bus controller. If a configuration file for an X2X module is stored on the bus controller, this module is configured accordingly and checked for correctness. As long as no special configuration is available, the module arrangement can be changed as required. If a module is required in a certain slot, the module type and configuration must be permanently stored in the defined slot in the form of a configuration file using method Apply.
- The configuration of X2X modules includes, for example, the desired sensor types for temperature modules, the channel configuration of analog modules, enabling status information or working in the packed format of the digital I/O modules.

As soon as a configuration has been successfully saved on the bus controller, it is retained even after a restart.

5.1 Configuration procedure

After switching on the power supply, the OPC UA bus controller determines all connected X2X modules, starts them automatically and creates an internal image of the input or output data. With this automatic configuration of the "supported modules" on page 12, the bus controller configures the I/O modules using the module configuration stored internally in the bus controller. As soon as configuration file "ConfigFile" exists, it is taken into account in this configuration.

Changing the configuration

- To configure the modules, the node provided on the bus controller (e.g. selections "DO/DI format packed", "StatusInfos", selection of sensor types) must be set accordingly.
- Configuration files can also be transferred from the OPC UA client to the OPC UA bus controller using bus controller method DownloadConfigurationFile.

Permanent application of the configuration

- Once the configuration has been completed, the modified configuration is entered in configuration file "ConfigFile" using method Apply and saved permanently.
- When saving for the first time using Apply, the connected module type is also entered in the configuration file and is required later in the corresponding slot. The defined module type can only be freed up again with a Delete procedure.
- After saving via Apply, the bus controller normally configures the corresponding module automatically. If necessary, it restarts the module to apply the configuration.

Restarting the bus controller

- If a restart of the bus controller is required after the configuration has been completed and saved, this request is displayed in the ConfigurationStatus.
- After deleting existing configuration files using method Delete or DeleteloModules, the bus controller must be restarted.
- During configuration (e.g. after disabling/enabling status inputs), it may be necessary to restart the bus controller in order to apply the configurations saved using Apply.

Information:

For unsupported X2X modules, the bus controller uses only default function model "254". A configuration file is not supported in this case. The configuration must be carried out again after each restart of the bus controller.
5.2 Configuration of the I/O modules

The I/O modules can be configured in 2 ways:

- Automatic configuration by the bus controller
- Manual configuration by the information model

5.2.1 Automatic configuration

After switching on the power supply, the OPC UA bus controller determines all connected I/O modules, starts them automatically and creates an internal image of the input or output data. With this automatic configuration of the "supported modules" on page 12, the bus controller configures the I/O modules using the module configuration stored internally in the bus controller. No information is requested from the modules regarding registers.

Information:

For unsupported X2X modules, the bus controller uses only default function model "254".

If an I/O module is missing or has a different hardware ID, a corresponding error is displayed as an LED status. See "LED status indicators" on page 6.

With method DownloadConfigurationFile, additional configurations generated in the application can be transferred to the bus controller.

5.2.2 Manual configuration

In addition to automatic configuration, the bus controller or I/O modules can also be configured manually.

I/O modules can be configured via the information model. The corresponding nodes are grouped in folder "Configuration", which exists for each module. The configuration parameters contained in this folder and the structure of these parameters are specific to each I/O module. For an explanation of parameters, see the respective module documentation. In addition to the configuration nodes, each configuration folder contains an object named "Control." The methods contained in this object are used to control the configuration process.

The current status of the configuration can be queried via node ConfigurationStatus.

Example of a module configuration

Information:

Modifying the configuration can lead to the addition or removal of configuration parameters. To detect this, the relevant higher-level node must be re-imported (browsed).

When method Apply is called, the changed configuration is saved permanently. A restart of the bus controller may also be required for the new configurations to be applied.
5.3 Unsupported modules

If a module is not listed in the list of "supported modules" on page 12, it is **not** supported. The bus controller starts up these unsupported modules with the module-specific default configuration and enters the module with the corresponding module ID (ModuleId) in the "SubDevices" list.

Configuration

The following conditions apply to each unsupported module:

- Modules start bus controller function model "254"
- Modified configurations must be enabled by calling `Apply`. Permanent storage does not take place.
- After each restart of the bus controller, the modules must be reconfigured.
- Neither data point "ConfigFile" nor method `DownloadConfigurationFile` are supported.
- Modules that are not supported can be configured via register access.
- For the names of the data and configuration registers, see the register documentation for the respective module.
- Register access to these modules is possible via methods `RegisterRead` and `RegisterWrite`.

5.4 Configuration files for modules

A separate configuration file in XML format can be created for each module slot and made available for access using OPC UA variable "ConfigFile".

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
<th>Keying</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigFile</td>
<td>XmlElement</td>
<td>utf-8</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / X2X / SubDevices / STx / ParameterSet&quot;</td>
</tr>
</tbody>
</table>

The XML configuration files can be downloaded from the bus controller using method `DownloadConfigurationFile`. A restart of the bus controller may be required to apply changes. To delete a configuration, the bus controller must always be restarted.

In addition, the configuration status should be evaluated during configuration processes. This displays the state of the respective configuration and any required restart of the bus controller.

If there is no or only the default configuration for a module, value "Null" is displayed instead of the configuration file:
Structure of the configuration file

If supported by the OPC UA client, the contents of a configuration file can be read out. The content of the XML file is displayed next to "Value" if one of the following conditions is met:

- The configuration was saved permanently using method Apply.
- The configuration was downloaded using bus controller method DownloadConfigurationFile and then applied with Apply.

encoding="utf-8": The XML file is stored in UTF-8 format.

Module DeviceId="7066": The module type is stored in this entry. This means that as soon as a configuration file for a slot is stored, a module with identical DeviceID must always be connected in this slot. If the DeviceId is different, the incorrectly connected module does not start up and an X2X Link error is indicated on the bus controller. For more information, see "Fixed-position modules" on page 50.

ID="ST3": To differentiate the configurations, the slot is also stored using ID="STx". This information also selects the associated slot when downloading the configuration file.

5.5 Configuration status

Configuration status "ConfigurationStatus" is available for each X2X module. In addition, the bus controller provides an overview of all statuses in the form of a status array.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationStatus</td>
<td>Int32 (Enum)</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / X2X / SubDevices / STx / ParameterSet&quot;</td>
</tr>
</tbody>
</table>

For more information, see "Evaluating the configuration status" on page 47 and "Detecting a necessary restart" on page 46.

5.6 Detecting a necessary restart

During configuration of the X2X modules, a restart of the OPC UA bus controller may be required for settings to be applied. This restart is unpredictable. For this reason, the ConfigurationStatus must always be evaluated in addition to the configuration.

A single restart after completion of all required configuration activities is sufficient, regardless of when or how often a necessary restart was reported.
5.7 Evaluating the configuration status

The configuration status can be queried via nodes **ConfigurationStatus** on the bus controller and each I/O module. This node indicates the current configuration state.

<table>
<thead>
<tr>
<th>Value</th>
<th>EnumString</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not configured</td>
<td>Not configured</td>
</tr>
<tr>
<td>1</td>
<td>No configuration available</td>
<td>A module for which no configuration has been saved is inserted in the slot. In this case, the module runs with its default configuration.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration ready</td>
<td>A configuration exists for a module in this slot, but the module is not available.</td>
</tr>
<tr>
<td>3</td>
<td>Configuration OK</td>
<td>The module is configured correctly.</td>
</tr>
<tr>
<td>4</td>
<td>Configuration changed but not applied</td>
<td>The configuration was changed. The modified parameters have not yet been applied or saved, however.</td>
</tr>
<tr>
<td>5</td>
<td>Module reboot triggered</td>
<td>A restart has been triggered for the module.</td>
</tr>
<tr>
<td>6</td>
<td>Wrong ModuleID</td>
<td>The module in the slot does not correspond to the module configured for this slot.</td>
</tr>
<tr>
<td>7</td>
<td>Reboot of bus controller required</td>
<td>The configuration was saved but could not be applied straightway. In this case, the saved configuration will be applied after the bus controller has restarted.</td>
</tr>
<tr>
<td>8</td>
<td>Error during configuration</td>
<td>An error occurred during configuration.</td>
</tr>
</tbody>
</table>

2 additional pieces of status information are available on the bus controller that represent the overall state of all configured X2X Link modules. This status information simplifies monitoring of the configuration status since only one node access is required to monitor the configuration of all X2X modules.

- **ConfigurationStatusAll**: General query due to a necessary restart of the bus controller.
- **ConfigurationStatusArray**: Individual configuration states of the X2X Link modules via array access.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationStatus</td>
<td>Int32 (Enum)</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / SubDevices / STx / ParameterSet&quot;</td>
</tr>
<tr>
<td>ConfigurationStatusAll</td>
<td>Int32 (Enum)</td>
<td>&quot;Root/Objects / DeviceSet / X20BC008U / ParameterSet&quot;</td>
</tr>
<tr>
<td>ConfigurationStatusArray</td>
<td>Int32 Array[201]</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / ParameterSet&quot;</td>
</tr>
</tbody>
</table>

1) The array offset corresponds to the slot.

**ConfigurationStatus**

The current configuration status of an individual module can be queried via node **Status / ConfigurationStatus** or directory **ParameterSet**.

The configuration status of the bus controller refers to the bus controller itself and usually has value "Configuration OK".

**ConfigurationStatusAll**

The entire configuration status can be queried via node **Status / ConfigurationStatusAll** or directory **ParameterSet**. This status is only available on the bus controller and contains the combined overall status of the bus controller and all configured X2X Link modules.

The individual **ConfigurationStatus** states have different values. Status "Reboot of bus controller required" always has the highest priority. As soon as a restart of the bus controller is required on at least one module, all other states are subordinate and only displayed on the corresponding modules themselves or in the **ConfigurationStatusArray**.
Bus controller configuration

Example

(3 connected modules and bus controller at offset 0)

The status of module 2 is returned although module 3 also requires application of the changed configuration by calling \texttt{Apply}.

<table>
<thead>
<tr>
<th>ConfigurationStatus</th>
<th>3 (Configuration OK)</th>
<th>Int32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationStatusAll</td>
<td>3 (Configuration OK)</td>
<td>Int32</td>
</tr>
<tr>
<td>ConfigurationStatusArray</td>
<td>3 (Configuration OK)</td>
<td>Int32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3 (Configuration OK)</td>
</tr>
<tr>
<td>1</td>
<td>1 (No configuration available)</td>
</tr>
<tr>
<td>2</td>
<td>7 (Reboot of buscontroller required)</td>
</tr>
<tr>
<td>3</td>
<td>1 (No configuration available)</td>
</tr>
<tr>
<td>4</td>
<td>0 (Not configured)</td>
</tr>
</tbody>
</table>

**Information:**

With the help of this node, a necessary restart of the bus controller can be easily detected.

Example

(3 connected modules and bus controller at offset 0)

If modules are not configured, the status of the bus controller is indicated as "Configuration OK" or "No configuration available".

<table>
<thead>
<tr>
<th>ConfigurationStatus</th>
<th>3 (Configuration OK)</th>
<th>Int32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationStatusAll</td>
<td>3 (Configuration OK)</td>
<td>Int32</td>
</tr>
<tr>
<td>ConfigurationStatusArray</td>
<td>3 (Configuration OK)</td>
<td>Int32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3 (Configuration OK)</td>
</tr>
<tr>
<td>1</td>
<td>1 (No configuration available)</td>
</tr>
<tr>
<td>2</td>
<td>1 (No configuration available)</td>
</tr>
<tr>
<td>3</td>
<td>1 (No configuration available)</td>
</tr>
<tr>
<td>4</td>
<td>0 (Not configured)</td>
</tr>
</tbody>
</table>

**ConfigurationStatusArray**

This node is only available on the bus controller and represents the configuration status of all available X2X Link modules in an array. The module slot corresponds to the array offset.

The configuration states of all modules can therefore be queried and evaluated with a single access to this array.
Example

(3 connected modules and bus controller at offset 0)

The modules in slots 2 and 3 indicate the changed but not yet applied configuration.

At the end of the entire configuration process, all changes can be accepted by calling method \texttt{ApplyAllConfigurations} one time.

To recognize all required \texttt{Apply} actions, the configuration status of each individual module should be evaluated. To permanently save a changed configuration, method \texttt{Apply} method must be executed. In addition, a restart of the bus controller may be required.

<table>
<thead>
<tr>
<th>Value</th>
<th>EnumString</th>
<th>Actions to be executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not configured</td>
<td>No action required</td>
</tr>
<tr>
<td>1</td>
<td>No configuration available</td>
<td>No action required</td>
</tr>
<tr>
<td>2</td>
<td>Configuration ready</td>
<td>Check why the module is malfunctioning (e.g. module not connected properly).</td>
</tr>
<tr>
<td>3</td>
<td>Configuration OK</td>
<td>No action required</td>
</tr>
<tr>
<td>4</td>
<td>Configuration changed but not applied</td>
<td>The changed configuration must be saved permanently using method \texttt{Apply}. Afterwards, a restart of the bus controller may be required to apply or enable the configuration.</td>
</tr>
<tr>
<td>5</td>
<td>Module reboot triggered</td>
<td>No action required, but wait until the X2X module has started up completely. The restart of the module was automatically initiated by the bus controller.</td>
</tr>
</tbody>
</table>
| 6     | Wrong ModuleId | Check the module configuration. Possible solutions:  
- Connect the properly configured module.  
- Delete an incorrect configuration. |
| 7     | Reboot of bus controller required | A single restart of the bus controller at the end of all configuration processes is sufficient. After receiving the command, the bus controller waits approx. 1 to 2 seconds until the restart is executed. During this time, the CPU should still free up all handles and close the connection to the bus controller.  
Options for restarting:  
- Calling method \texttt{Reboot}  
- Switching the power supply of the bus controller off and on  
- Call methods \texttt{UA\_MethodGetHandle} and \texttt{UA\_MethodCall} in function blocks |
| 8     | Error during configuration | Repeat or delete the configuration. |

Information:

All connections established with \texttt{UA\_Connect()} must be closed with \texttt{UA\_Disconnect()} before the bus controller or CPU is restarted.
5.8 Fixed-position modules

The bus controller automatically configures all I/O modules connected on the X2X Link network and enters them in the "SubDevices" list. The module arrangement is freely selectable and can be changed as needed.

If a module is required in a specific slot, its module type can be permanently stored in the slot using method Apply. The configuration of the module type is also stored in the respective slot.

As soon as a certain module type is configured to a slot, a module of another type connected at this position no longer functions without errors. An incorrectly connected module has the following effects:

- The module is indicated by LED "r" flashing green and remains in mode PREOPERATIONAL.
- A corresponding error message is entered in "ConfigurationStatus".
- The desired module type is displayed in "ConfigFile".
- The red "MS" error LED on the bus controller blinks.

Information:
The set configuration is permanently applied using method Apply. Modules of another type on the stored slot result in an error message.

Calling the method may require a restart of the bus controller (see "Execution status of "Apply" methods" on page 55). A restart after all changes have been made is sufficient.

Unlocking and freeing up fixed-position modules

Saved configurations can be deleted and thus the slot can be freed up for another module type using method Delete.

Information:
Method "Delete" always requires a subsequent restart of the bus controller in order to carry out the deletion process correctly.

5.9 Deleting an existing configuration

- The configuration of the selected module can be deleted by calling method Delete followed by a restart of the bus controller.

Information:
Method "Delete" is not permitted to be applied with ApplyChanges since this call saves and fixes the currently connected modules in a default configuration file.

- With bus controller firmware versions >1.06, all X2X module configurations can be deleted using method DeleteloModules with a single call of method Call.
- By resetting to factory settings (see "Restoring to factory settings" on page 10), the configurations of all modules are deleted and all internal structures of the bus controller are reset with default values.
6 X2X module status

In addition to the configuration status information, the state of the X2X Link network and modules is important. The exchange of I/O data via the X2X Link network and the startup of X2X modules is performed independently by the OPC UA bus controller.

6.1 ModuleOK

For each X2X module, the ModuleOk status can be queried via node "ModuleOk". Status ModuleOk indicates the correct operating state of the corresponding module.

A status array with all ModuleOk information is available on the bus controller.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Data type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModuleOkArray</td>
<td>Boolean Array[201] 1)</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / ParameterSet&quot;</td>
</tr>
<tr>
<td>ModuleOk</td>
<td>Boolean</td>
<td>&quot;Root / Objects / DeviceSet / X20BC008U / SubDevices / Tx / ParameterSet&quot;</td>
</tr>
</tbody>
</table>

1) The array offset corresponds to the slot.

Conditions for ModuleOk flag values

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TRUE</td>
<td>• The X2X module must be fully uploaded (mode OPERATIONAL).&lt;br&gt;• The X2X module must signal the validity of the data.</td>
</tr>
<tr>
<td>0</td>
<td>FALSE</td>
<td>• The X2X module does not provide new data for a certain time.&lt;br&gt;• The X2X module is detected as failed by the bus controller.&lt;br&gt;• The X2X module is not fully operational (not in mode OPERATIONAL).&lt;br&gt;• The X2X module no longer supplies valid data.</td>
</tr>
</tbody>
</table>

Information:

The input data is only valid if flag ModuleOk is set to TRUE and there is a correct connection to the OPC UA bus controller.

6.2 NetworkStatus

The network status provides information about the operating state of the X2X Link stations, i.e. the bus modules of the respective I/O modules.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I/O bus supply, 1 = OK</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>I/O bus, 1 = OK</td>
</tr>
<tr>
<td>3</td>
<td>DataValid, 0 = OK, 1 = Old data</td>
</tr>
<tr>
<td>4 - 7</td>
<td>Always 1</td>
</tr>
</tbody>
</table>

This results in the following values:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value (decimal)</th>
<th>Value (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X2X Link station inactive (e.g. no X2X Link power supply)</td>
<td>0</td>
<td>0x00</td>
</tr>
<tr>
<td>Everything OK (I/O data valid)</td>
<td>245</td>
<td>0xF5</td>
</tr>
<tr>
<td>No communication with the electronics module (bits 7 to 3 invalid)</td>
<td>249</td>
<td>0xF9</td>
</tr>
<tr>
<td>I/O data invalid; communication between X2X ASIC and electronics module OK (ASIC carried out a valid &quot;Sync in&quot; transfer with the electronics module in the previous X2X Link cycle)</td>
<td>253</td>
<td>0xFD</td>
</tr>
</tbody>
</table>

Information:

Any value unequal to 245 (0xF5) means that the I/O data of the corresponding module is invalid. This circumstance is usually taken into account or processed accordingly in the application.
7 Methods

The methods supported by the bus controller are listed and can be enabled individually for each module. Methods could be enabled via various subnodes. All available methods of the module are listed collectively in node "MethodSet".

<table>
<thead>
<tr>
<th>Only available for bus controllers</th>
<th>Available for each module</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; MethodSet</td>
<td>&gt; MethodSet</td>
</tr>
<tr>
<td>&gt; ApplyAllConfigurations</td>
<td>&gt; Apply</td>
</tr>
<tr>
<td>&gt; ApplyChanges</td>
<td>&gt; Delete</td>
</tr>
<tr>
<td>&gt; Delete</td>
<td>&gt; LoadDefault</td>
</tr>
<tr>
<td>&gt; DeleteLoModules</td>
<td>&gt; RegisterRead</td>
</tr>
<tr>
<td>&gt; DownloadConfigurationFile</td>
<td>&gt; RegisterWrite</td>
</tr>
<tr>
<td>&gt; LoadDefault</td>
<td>&gt; Restore</td>
</tr>
<tr>
<td>&gt; Reboot</td>
<td></td>
</tr>
<tr>
<td>&gt; Restore</td>
<td></td>
</tr>
</tbody>
</table>
7.1 Calling methods

The methods provided by the bus controller on the bus controller or X2X modules can in principle be executed by any OPC UA client that supports this functionality.

- Calling methods with UaExpert
- Executing methods using library "AsOpcUac"

7.1.1 Calling methods with UaExpert

In UaExpert, all methods can be executed by right-clicking on the method and clicking on "Call".

All arguments can be added in the dialog window and executed by clicking on the "Call" button.

Input arguments

Input arguments are optional parameters passed to the method and only displayed if necessary. If one or more input arguments are required, the input option is displayed automatically.

Output arguments

Output arguments are optional parameters returned by the method and only displayed if necessary. If one or more output arguments are required, they are displayed automatically.

Result

Each time a method is called using "Call", the execution status is returned in the call window under "Result". If execution is successful, "Succeeded" is displayed; an error message is displayed in the event of an error.
Methods

7.1.2 Executing methods using library "AsOpcUac"

Methods of the OPC UA bus controller can be executed using library "AsOpcUac".

● Necessary function block calls are:

<table>
<thead>
<tr>
<th>Function Block Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UA_MethodGetHandle()</td>
<td>Reserves an access handle</td>
</tr>
<tr>
<td>UA_MethodCall()</td>
<td>Executes the method using the reserved handle</td>
</tr>
<tr>
<td>UA_MethodReleaseHandle()</td>
<td>Frees up the reserved handle</td>
</tr>
</tbody>
</table>

● When UA_MethodGetHandle() is called, all information about the required input and output arguments with names and DataType is additionally loaded from the OPC UA bus controller.

● When UA_MethodCall() is called, the validity of the specified arguments is checked first, e.g. completeness, argument name, matching data type. Only then is the actual method executed on the server.

● Passed and returned arguments

All passed and returned arguments are of type UAMethodArgument. For more information, see "Details about DataType ExtensionObject “UAMethodArgument”" on page 29.

For information on determining the required data type, see "Determining DataType for method calls" on page 27.

Input arguments

The input arguments for the method only have to be defined if required, depending on the method called. If input arguments are not required, value NULL is specified. If multiple input arguments are required, they must be specified in the correct order.

Output arguments

The output arguments from the method only have to be defined if required, depending on the method called. If output arguments are not required, value NULL is specified. If multiple output arguments are required, they must be specified in the correct order.

Result

The output argument only needs to be defined if necessary. If output arguments are not required, value NULL is specified.
7.2 Execution status of "Apply" methods

After calling a method using "Call", the configuration result is returned in the call window under "Output arguments".

<table>
<thead>
<tr>
<th>Value</th>
<th>EnumString</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not configured</td>
<td>Not configured</td>
</tr>
<tr>
<td>1</td>
<td>No configuration available</td>
<td>A module for which no configuration has been saved is inserted in the slot. In this case, the module runs with its default configuration.</td>
</tr>
<tr>
<td>2</td>
<td>Configuration ready</td>
<td>A configuration exists for a module in this slot, but the module is not available.</td>
</tr>
<tr>
<td>3</td>
<td>Configuration OK</td>
<td>The module is configured correctly.</td>
</tr>
<tr>
<td>4</td>
<td>Configuration changed but not applied</td>
<td>The configuration was changed. The modified parameters have not yet been applied or saved, however.</td>
</tr>
<tr>
<td>5</td>
<td>Module reboot triggered</td>
<td>A restart has been triggered for the module.</td>
</tr>
<tr>
<td>6</td>
<td>Wrong ModuleId</td>
<td>The module in the slot does not correspond to the module configured for this slot.</td>
</tr>
<tr>
<td>7</td>
<td>Reboot of bus controller required</td>
<td>The configuration was saved but could not be applied straightaway. In this case, the saved configuration will be applied after the bus controller has restarted.</td>
</tr>
<tr>
<td>8</td>
<td>Error during configuration</td>
<td>An error occurred during configuration.</td>
</tr>
</tbody>
</table>

Some configuration results require further action.

<table>
<thead>
<tr>
<th>Value</th>
<th>EnumString</th>
<th>Actions to be executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not configured</td>
<td>No action required</td>
</tr>
<tr>
<td>1</td>
<td>No configuration available</td>
<td>No action required</td>
</tr>
<tr>
<td>2</td>
<td>Configuration ready</td>
<td>Check why the module is malfunctioning (e.g. module not connected properly).</td>
</tr>
<tr>
<td>3</td>
<td>Configuration OK</td>
<td>No action required</td>
</tr>
<tr>
<td>4</td>
<td>Configuration changed but not applied</td>
<td>The changed configuration must be saved permanently using method Apply. Afterwards, a restart of the bus controller may be required to apply or enable the configuration.</td>
</tr>
<tr>
<td>5</td>
<td>Module reboot triggered</td>
<td>No action required, but wait until the X2X module has started up completely. The restart of the module was automatically initiated by the bus controller.</td>
</tr>
</tbody>
</table>
| 6     | Wrong ModuleId | Check the module configuration. Possible solutions:  
  - Connect the properly configured module.  
  - Delete an incorrect configuration. |
| 7     | Reboot of bus controller required | A single restart of the bus controller at the end of all configuration processes is sufficient. After receiving the command, the bus controller waits approx. 1 to 2 seconds until the restart is executed. During this time, the CPU should still free up all handles and close the connection to the bus controller. Options for restarting:  
  - Calling method Reboot  
  - Switching the power supply of the bus controller off and on  
  - Call methods UA_MethodGetHandle and UA_MethodCall in function blocks  
  All connections established with UA_Connect() must be closed with UA_Disconnect() before the bus controller or CPU is restarted. |
| 8     | Error during configuration | Repeat or delete the configuration. |
7.3 Supported methods

7.3.1 Apply

There are 2 different methods for saving changed values.

- ApplyChanges - Save a single value
- ApplyAllConfigurations - Save all values

7.3.1.1 ApplyChanges

Changed values are only saved by calling method "ApplyChanges". The method has an output argument that indicates the configuration status at the end of the call.

If the new configuration leads to the creation of additional cyclic data on the X2X Link network, the new parameters will not be applied immediately. In this case, the bus controller must be restarted (see "Reboot" on page 61). This is indicated by configuration status "Reboot of bus controller required". The bus controller does not have to be restarted immediately, however. Further I/O modules can be configured first.

If the module has been reconfigured without increasing the size of the cyclic X2X frame, only the I/O module is restarted. This is indicated by configuration status "Module reboot triggered". After restarting the module, the changed configuration was applied. This is indicated by configuration status "Configuration OK".

As soon as method "ApplyChanges" has been called, the module is permanently assigned to a slot. From then on, the slot will only accept modules of the same type. If another module type is connected, this module does not boot up completely and remains in mode PREOPERATIONAL mode. Configuration status "Wrong ModuleId" is displayed.

7.3.1.2 ApplyAllConfigurations

This method is in the MethodSet of the X20BC008U object. This means that all changed configurations can be applied at once. This replaces individual calls of method ApplyChanges.

When method "ApplyAllConfigurations" is called, all changed configurations are initially saved. Then it is checked whether one of the configuration changes requires a restart of the bus controller. If so, this is indicated in the corresponding output argument for the method and the procedure is ended. The configuration is only applied after the bus controller has been restarted, e.g. by calling method Reboot. If no restart of the bus controller is required, all I/O modules requiring a restart due to the configuration change are restarted automatically.

7.3.2 Delete

When method "Delete" is called, the configuration file for the corresponding module is deleted. However, the currently active configuration is not modified.

Modules of any type can be connected in the slot used. After the bus controller has restarted, the module that is currently inserted is started with its default configuration.

7.3.3 DeleteIoModules

This method deletes the configuration files of all configured X2X modules.

This method is only available with firmware versions ≥V1.13.

Information:

Each deletion of a configuration, i.e. both "Delete" and "DeleteIoModules", requires a restart of the bus controller.

7.3.4 DownloadConfigurationFile

This method allows configuration data to be downloaded to the bus controller in XML format. For more information, see "Configuration files for modules" on page 45.

7.3.5 LoadDefault

This method loads the default configuration for the relevant module. All parameters are reset to their default setting. However, the configuration is only saved and transferred when method Apply is called.
7.3.6 OutputKeepAlive

This method defines the behavior of the digital outputs after connection loss. By default, the digital outputs maintain the current output states after connection loss. This means that outputs remain set if they were set.

This method is only available with firmware versions V1.18 or later.

**Enabling mode “OutputKeepAlive”**

- Method "OutputKeepAlive" must be enabled with at least 1 client.
- Configure parameters OutputKeepAliveMode and OutputKeepAliveTime if they have not been saved in flash memory.
  For more information, see "Configuration" on page 58.

**Information:**

The value must be ≠ 0 for both parameters.

- Call method "OutputKeepAlive".
- Status parameter OutputKeepAliveStatus is set to "1 OutputKeepAlive enabled".
  For more information, see "Status information" on page 59.

**Procedure**

The following diagram shows the error-free flow of the method call.

For the behavior in the event of error, see "Behavior in the event of error" on page 60.

**Information:**

- Call method "OutputKeepAlive" cyclically before the time specified in parameter OutputKeepAliveTime expires. This method can also be called by different masters.

- Each call resets the internal counter value to the time specified in OutputKeepAliveTime.
- Status registers except OutputKeepAliveTimeRemaining remain unchanged.
  For evaluation of status registers, see "Status information" on page 59.
- Any access to the digital outputs is possible.
Disabling mode "KeepAlive"

Mode "KeepAlive" can be disabled in the following ways:

- By restarting the bus controller.
- Setting parameter OutputKeepAliveTime and/or OutputKeepAliveMode to 0.

For more information, see "Configuration" on page 58.

Status parameter OutputKeepAliveCounter is reset to value 0.

7.3.6.1 Configuration

All configurations of the method can be made in folder ". / Status / OutputKeepAlive" using 2 configuration parameters.

After calling method OutputKeepAlive for the first time, every change of a parameter is applied immediately, even without permanent storing.

The configuration parameters are not stored permanently in flash memory until method Apply is called. As soon as a value is stored in flash memory, it will be used after the bus controller is started. Otherwise, the parameters will be initialized with value 0.

Parameter "OutputKeepAliveMode"

This parameter is of data type enum (DINT, int32).

<table>
<thead>
<tr>
<th>Values</th>
<th>Mode</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Compatibility mode</td>
<td>In the event of connection loss, the current switching states of the outputs are retained.</td>
</tr>
<tr>
<td>1</td>
<td>Output ResetsToZero</td>
<td>In the event of connection loss, the output states of all outputs are set to 0.</td>
</tr>
</tbody>
</table>

Parameter "OutputKeepAliveTime"

This parameter contains the time in milliseconds within which method OutputKeepAlive must be called again. An error state is triggered after this time has expired.

The time required for executing this method is a multiple of the X2X Link cycle time of 2 ms. Values should therefore be given in 2-ms steps.

**Information:**

Changing this parameter while method "OutputKeepAlive" is enabled triggers an error state if the newly defined time has already elapsed since the last method call.
7.3.6.2 Status information

All status information of the method are available in folder ". / Status / OutputKeepAlive" as 3 status parameters.

Parameter "OutputKeepAliveCounter"

This parameter indicates the number of fault events that occur. Each time "OutputKeepAliveStatus" = "Error OutputKeepAlive active", this counter is incremented by one.

Parameter "OutputKeepAliveStatus"

This parameter returns the following statuses:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OutputKeepAlive disabled</td>
<td>OutputKeepAlive disabled</td>
</tr>
<tr>
<td>1</td>
<td>OutputKeepAlive enabled</td>
<td>OutputKeepAlive enabled</td>
</tr>
<tr>
<td>2</td>
<td>Error OutputKeepAlive active</td>
<td>Indicates an active error state.</td>
</tr>
</tbody>
</table>

In addition, the error state is indicated by 3 pulses on the red "CS" LED.

Parameter "OutputKeepAliveTimeRemaining"

This parameter specifies the remaining time in milliseconds until a timeout occurs. An error state is triggered after time has expired.

With each call of method OutputKeepAlive, this parameter is reset to the value of parameter OutputKeepAliveTime.

Time value 0 means:

- KeepAlive mode is not active.
- Or: The time has already expired.
7.3.6.3 Behavior in the event of error

The following diagram shows an error that is caused by method calls that were not executed in time.

- After a method call, the method must be called again within the time set in parameter `OutputKeepAliveTime`.
- After time has expired, an error state is triggered as soon as status parameter `OutputKeepAliveTimeRemaining` indicates value 0.
  - Status register `OutputKeepAliveStatus` is set to value "2 Error OutputKeepAlive active"
  - Status counter `OutputKeepAliveCounter` is increased by 1.
  - All digital outputs are set to ZERO and locked.
- Error states are lifted when calling method `OutputKeepAlive` again. The lock is lifted and status parameter "OutputKeepAliveStatus" is reset.

Locking the digital outputs

The following must be taken into account during an error state:

- Write access is no longer possible. Each access is rejected with error code UA_SCBADNOTSUPPORTED (0x803D0000).
- Read access to the digital outputs is possible.
- Write access is displayed as an error message in the log area.

Log area of UaExpert

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Source</th>
<th>Server</th>
<th>Message</th>
</tr>
</thead>
</table>

Log area of Automation Studio

<table>
<thead>
<tr>
<th>Sever</th>
<th>Time</th>
<th>ID</th>
<th>Area</th>
<th>Entered</th>
<th>Add new filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>2018-12-18 17:11:52.137</td>
<td>21</td>
<td>BAR</td>
<td>d6Op0C102Device</td>
<td>Write node,0,Objects,2,DeviceData,m.SetDigitalOutput2.value failed E=603D0000</td>
</tr>
<tr>
<td>Warning</td>
<td>2018-12-18 17:11:52.137</td>
<td>21</td>
<td>BAR</td>
<td>d6Op0C102Device</td>
<td>Write node,0,Objects,2,DeviceData,m.SetDigitalOutput2.value failed E=603D0000</td>
</tr>
</tbody>
</table>
7.3.7 RegisterRead

This method is located in the method set of the I/O module object. It allows I/O module registers to be read directly. As an input argument, this method requires the number corresponding to the register that is to be read. The read value is returned as an output argument.

7.3.8 Reboot

This method is in the method set of the X20BC008U object. It allows the bus controller to be restarted. If the method is called, all connected clients are disconnected and the bus controller and all I/O modules are restarted.

Starting with firmware version V1.13, it is possible to restart each connected X2X module individually by calling a method. This is normally not required since the bus controller automatically restarts the connected X2X modules if necessary. In addition, the NodeID of the reboot method has been modified.

- NodeID in firmware <V1.13: 120
- NodeID in firmware ≥V1.13: 240

7.3.9 RegisterWrite

This method is located in the method set of the I/O module object. It is used to write registers to the relevant I/O module directly. The register number and the value to be written are transferred as input arguments.

7.3.10 Restore

Calling this method restores the last saved configuration.
8 OpcUa_any driver as client application

As an alternative to controlling OPC UA bus controller X20BC008U via library AsOpcUac, it is also possible to control it via the OpcUa_any driver. This driver is available for controlling any OPC UA server. A generic standard configuration is created on bus controller X20BC008U for the connected modules. This eliminates the need of additional programming when working with the standard configuration.

In Automation Studio, the OpcUa_any driver is added in the "Physical View" and configured with the desired parameters.

**Example**
Configuration with a local network: IP address 192.168.1.2 Port 4840

- **Information:**
  Server bus controller X20BC008U only permits security mode (UASecurityMsgMode_None) without user and password query.

**Configuration of data points**

All desired data points must be configured as channels.

- **Information:**
  Only standard data types such as BOOL, (U)SINT, (U)INT and (U)DINT can be accessed. String variables or more complex variable types are not supported by the OpcUa_any driver control.
9 Getting started

The following material is sufficient for the first steps:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X20BB80</td>
<td>Bus module for bus controller and power supply module</td>
</tr>
<tr>
<td>X20BC008U</td>
<td>OPC UA server bus controller</td>
</tr>
<tr>
<td>X20PS9400</td>
<td>Power supply module</td>
</tr>
<tr>
<td>X20TB12</td>
<td>Terminal block for wiring the X20 power supply on the power supply module</td>
</tr>
<tr>
<td>Cables</td>
<td>RJ45 network patch cables</td>
</tr>
<tr>
<td>&quot;Optional&quot;</td>
<td>X20 input and output modules with backplane modules (from the list of &quot;supported modules&quot; on page 12)</td>
</tr>
</tbody>
</table>

9.1 Configuring the network

To gain initial experience with OPC UA, it is advantageous to operate the bus controller in a local network. After connecting the bus controller to the PC via Ethernet cable, the IP address must be set for all devices.

9.1.1 Setting the IP address on the bus controller

The bus controller is set to address 192.168.1.1 by default. If necessary, this IP address can be changed using the address switches. For more information, see "Overview of network address switch functionality" on page 7.

9.1.2 Setting the IP address on the PC

The IP address on the PC must be configured to the same subnet as the bus controller, e.g. 192.168.1.100.

Double-click on "Internet Protocol". Add the IP address.
9.1.3 Automation Studio IP address

If a CPU is used to connect to the bus controller via the OPC UA client library, its IP address must also be changed to an address in the same subnet. The address can be set in Automation Studio under "Physical View → ETH".

9.1.4 Automatic configuration via DHCP

When operating the bus controller in a network with a DHCP server, the address can be obtained automatically.

- On the bus controller: Set parameter "X20BC008U / Configuration / Network / EnableDHCP" to TRUE.
- Operate the bus controller with address switch position 0x80 to 0xEF. As long as no address has been assigned by the DHCP server, the LED status indicator "CS" blinks red.
- Assign a unique hostname. (See "Automatic IP assignment by a DHCP server" on page 8.)

9.2 Checking the configuration

Commands "ipconfig" and "ping" in the PC command prompt are effective tools for commissioning or troubleshooting networks.

- The interface configuration of the PC can be checked using "Ipconfig".
- "Ping" can be used to check whether the target station is accessible in the network. When using a hostname, the IP address is also displayed.
9.3 OPC UA client freeware tool UaExpert

After network configuration, freeware tool UaExpert from Unified Automation can be used. Due to the graphical representation of object structures, this tool is well suited for gaining initial OPC UA experience.

9.3.1 Connecting with UaExpert

This example shows how a bus controller can be connected to OPC UA client software UaExpert.

1) Click on "Add server" to add a server.
2) Double-click on "<Double-click to add server>" to enter the server's URL. Example: "opc.tcp://bropc215:4840" or "opc.tcp://192.168.1.1:4840"
3) If the server can be reached over the network, double-click on "None - None (uatcp-uasc-uabinary)" to add it to the server list in the main window.
4) Click on "Connect server" to establish the connection.

As soon as the connection has been established, the structure of the "address space" provided by the bus controller will be shown in its own window. The bus controller is displayed under "DeviceSet" and contains all objects provided by the bus controller.
9.3.2 Show IP address

To check the IP address assigned by the DHCP server, the connection to the bus controller must be established in UaExpert via the hostname and port number:

Example: "opc.tcp://X20BC008U:4840".

The address assigned by the DHCP server can then be queried in object "CurrentIP-Address".
9.3.3 Displaying and changing hostnames

If several bus controllers are used in the same network, the connection must be established via a unique hostname. The hostname is identical by default for all bus controllers even if different IP addresses are used. To change the host name, select parameter "Hostname" under "Configuration → Network". Under "Value", the hostname can be changed and saved after right-clicking and selecting Apply. To permanently apply this setting, the bus controller must be restarted.

Under "Properties", the hostname can be displayed and changed.

9.4 Firmware update

A necessary firmware update can be performed using the built-in web server. For more information, see "Web server" on page 11.

9.5 Restarting the bus controller

If necessary, the bus controller must now be restarted.
10 Programming example

The examples presented in this section show proposed solutions to frequently required tasks. All examples were programmed in Structured Text in Automation Studio. Read and write accesses, method calls and configurations, etc. are performed using OPC UA library "AsOpcUac".

This section does not contain any basic or additional information. For a detailed explanation, see "Basic information" on page 14 and "Bus controller objects and directories " on page 34.

Information:
Library "AsOpcUac" is required to implement the examples. This is supported in Automation Runtime V4.10 and later.

The following requirements apply to all examples:

• All programming examples are developed in Automation Studio.
• An established connection to the bus controller is required to exchange data and execute services. A maximum of 5 parallel connections can be established per bus controller.
• The connection is established using UA_Connect(). It is not repeated for every example.
• All connections established with UA_Connect() must be freed up again with UA_Disconnect().
• All handles reserved with UA_NodeGetHandle() must be freed up again with UA_NodeReleaseHandle().
• To determine the NodeIDs using UA_TranslatePath(), the NamespaceIndex should be read using UA_GetNamespaceIndex() and taken into account for generating the RelativePath name.
• The names and NodeIDs from NamespaceIndex 0 remain unchanged by default and are specified as StartNodeID.
• The display of the correct structure of the relative path names is partially omitted in the programming examples.

10.1 Preparing the OPC UA client

The OPC UA system is enabled in the PLC configuration.

Parameter "Activate OPC UA system" is set to "On", and the following parameters are then changed:

• Port number = "4840"
• Auditing Server Facet = "Off" (The OPC UA server bus controller does not support audit events.)
• Security policies / No security = "Prohibit"
• Security admin = "Administrator"
10.2 Function blocks - General information

10.2.1 Execute-Done model

The function blocks in the library work according to the "Execute-Done" model.

The following diagram shows possible states of "Execute" as well as status outputs "Busy", "Done" and "Error".

Execution of the block begins on a rising edge of input "Execute". The block indicates active processing by setting output "Busy". With asynchronous blocks, processing can take several cycles.

If execution is ended, the busy output is deleted. The error output indicates whether an error occurred during execution (see "Error handling" on page 70):

- Diagram case 1: An error occurred during execution.
- Diagram case 2: Execution was successful.

You can delete input "Execute" while processing is active. In this case, status outputs "Done" and "Error" are only valid for one cycle. The execution of the block is not aborted prematurely.

Status outputs "Busy", "Done" and "Error" are mutually exclusive. Only one of these outputs can be set at a time.
10.2.2 Error handling

If an error occurs during execution of a function block, output "Error" is set and the corresponding error number (or list of error numbers) can be evaluated on output "ErrorID" (or "ErrorIDs").

Connection error

Error handling begins by assessing whether an interruption in communication has occurred. An aborted connection is indicated by error IDs "Bad_Disconnect" (0x80AD0000) or "Bad_Timeout" (0x800A0000).

In this case, the OPC UA system tries to re-establish the connection. Any data to be transmitted, such as a subscription, is invalid from this point in time. The client can wait for the new connection to be established, e.g. by repeatedly calling the last function block or evaluating the connection status. In these cases, Bad_Disconnect is returned until the aborted connection is corrected. All previously determined handles are then valid again, and the normal program sequence can be resumed.

Other errors

For other errors, you must decide how to handle the error according to the error number.

For detailed error and status descriptions, see Automation Help.

10.2.3 Read and write operations

Before performing read or write operations

Before objects can be read or written to, several preparatory function block calls are required.

- Establish a connection to the bus controller with function block UA_Connect().
- Determine the NodeIDs using the specified directory or object data with function block UA_TranslatePath(). These NodeIDs are the basic requirement for all subsequent accesses to objects or methods.
- Reserved the handle for accessing a directory or object via its NodeID with function block UA_NodeGetHandle() or UA_NodeGetHandleList. This handle is then retained throughout the entire session.

Information:

For read or write access, matching process variables must be created in the data type in the Automation Studio program. For more information, see "Node identification" on page 31.

Any number of read and write calls can now be made.

After the last call of the read or write operation

- Free up the handles with function block UA_NodeReleaseHandle().
10.3 Function blocks - Application

The following table provides an overview of the most important function blocks for OPC UA programming and their use.

<table>
<thead>
<tr>
<th>Function block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UA_Connect()</td>
<td>Establish connection to the OPC UA bus controller.</td>
</tr>
<tr>
<td>UA_Disconnect()</td>
<td>Disconnect the OPC UA bus controller.</td>
</tr>
<tr>
<td>UA_ConnectionGetStatus()</td>
<td>Query the connection status to the OPC UA bus controller.</td>
</tr>
<tr>
<td>UA_GetNamespaceIndex()</td>
<td>Query the used Namespacelndex of the OPC UA bus controller.</td>
</tr>
<tr>
<td>UA_TranslatePath()</td>
<td>NodeID of an object or directory.</td>
</tr>
<tr>
<td>UA_NodeGetHandle()</td>
<td>Reserve handle to NodeID of an object or directory.</td>
</tr>
<tr>
<td>UA_NodeReleaseHandle()</td>
<td>Free up handle to NodeID of an object or directory.</td>
</tr>
<tr>
<td>UA_Read()</td>
<td>Read object information.</td>
</tr>
<tr>
<td>UA_Write()</td>
<td>Write object information.</td>
</tr>
<tr>
<td>UA_NodeGetHandleList()</td>
<td>Determine the NodeID of a list object.</td>
</tr>
<tr>
<td>UA_ReadList()</td>
<td>Read object information from a list.</td>
</tr>
</tbody>
</table>

10.3.1 UA_Connect()

This function block establishes and maintains (active) the connection to the OPC UA bus controller. It remains active until UA_Disconnect() is performed or the connection is disconnected mechanically, e.g. by disconnecting the connection cable.

Caution!

If the OPC UA connection between the client and bus controller is lost, the current module states are frozen. Outputs are not reset.

- The OPC UA bus controller does not yet support security functionalities. Only a connection of type "UserIdentityTokenType := UAUITT_Anonymous" can be established.
- All connections established with UA_Connect() must be freed up again with UA_Disconnect().
- A maximum of 5 parallel connections to the OPC UA bus controller can be established. If active connections are not closed with UA_Disconnect(), they still occupy one of the available 5 connections.
- Connections that are not freed up are only freed up again after restarting the bus controller or by using UA_Disconnect() on the active connection.
- The function block tries to maintain or automatically re-establish the connection, e.g. after a bus controller failure. In this case, the function block returns a new connection handle and the application continues to run.
Application tips

In order to establish a connection via the hostname, parameter "DNS" must be enabled accordingly in the CPU configuration (see "Connection management" on page 14).

When a new connection is established, the newly generated connection handle "ConnectionHdl" on the bus controller is usually different from the previous handle. However, it can also be identical. For this reason, do not check for a different handle to determine whether the connection is established.

```pascal
(*----------------------------------------------------*)
(* UA_Connect - establish connection to OPC UA Server *)
(*----------------------------------------------------*)
SessionConnectInfo_0.SecurityMsgMode   := UASecurityMsgMode_None;
SessionConnectInfo_0.SecurityPolicy    := UASecurityPolicy_None;
SessionConnectInfo_0.TransportProfile  := UATP_UATcp;
SessionConnectInfo_0.UserIdentityToken.UserIdentityTokenType := UAUITT_Anonymous;
SessionConnectInfo_0.SessionTimeout    := T#1m;
SessionConnectInfo_0.MonitorConnection := T#10s;
ServerEndpointUrl   := 'opc.tcp://192.168.1.5:4840';    // BC Station #5

(*----------------------------------------------------*)
(*-- UA_Connect --*)
(*----------------------------------------------------*)
UA_Connect_0(Execute            := ExecuteConnect_0,
             SessionConnectInfo := SessionConnectInfo_0,
             Timeout            := T#10s);
IF (UA_Connect_0.Done = TRUE) THEN
  //==> Connection is activated and OK
  (*-------- Parameter for next function block --------*)
  ConnectionHdl := UA_Connect_0.ConnectionHdl; // -> for all further FB calls needed
  (*---------------------------------------------------*)
ELSIF (UA_Connect_0.Error = 1) THEN
  ConnectionHdl := 0;     // -> old Handle not valid anymore
END_IF
```
10.3.2 UA_Disconnect()

This function block closes the active connection to the OPC UA bus controller and frees up the connection again on the bus controller.

- UA_Disconnect() can only be performed on an active connection.
- UA_Disconnect() frees up one of the 5 possible connections on the bus controller. This can be used again to establish another connection.

Application tip

If UA_Disconnect() is required due to method Reboot, it is necessary to wait 1 to 2 seconds so that the bus controller is guaranteed to be in REBOOT mode. Otherwise, the connection will be re-enabled by mistake before the actual restart. After this time, enabling of function block UA_Connect() must be FALSE, i.e. "Disabled", for at least one program cycle before the connection can be re-established.

```
(*-------------------------------------------------------*)
(*-- UA_Disconnect --                                 *)
(*-------------------------------------------------------*)
UA_Disconnect_0(Execute       := ExecuteDisconnect_0,
                ConnectionHdl := ConnectionHdl,
                Timeout       := T#10s);
IF (UA_Disconnect_0.Busy = 0) THEN
  IF (ExecuteDisconnect_0 = TRUE) THEN
    ExecuteDisconnect_0  := FALSE;  //-> Release the FB for further use
    ConnectionHdl        := 0;      // -> old Handle not valid anymore
    //--> if BC Reboot wait 5 sec after Disconnect -> Then Reconnect
  END_IF
END_IF
```

10.3.3 UA_ConnectionGetStatus()

This function block can be used to query the current connection status or to monitor the validity of the connection. Further required actions can be derived from the returned connection status.

Enum constants for the connection status

<table>
<thead>
<tr>
<th>Enum constants</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UACS_Connected</td>
<td>0</td>
<td>The UA client is connected to the UA server.</td>
</tr>
<tr>
<td>UACS_ConnectionError</td>
<td>1</td>
<td>The connection between the UA client and UA server is faulty.</td>
</tr>
<tr>
<td>UACS_Shutdown</td>
<td>2</td>
<td>The connection between the UA client and UA server is interrupted.</td>
</tr>
</tbody>
</table>
Application tip

The connection status can be queried both cyclically and only when necessary. After a restart of the bus controller or UA_Disconnect(), it can be determined in a time-optimized manner from when error-free data access to the bus controller can take place again.

```plaintext
(*----------------------------------------------------*)
(*-- Check connection - until BC is connected --*)
(*----------------------------------------------------*)
UA_ConnectionGetStatus_1(Execute := TRUE,
  ConnectionHdl := ConnectionHdl,
  Timeout := T#2s);

IF (UA_ConnectionGetStatus_1.Busy = 0) THEN
  iErrorID := UA_ConnectionGetStatus_1.ErrorID;
  IF (UA_ConnectionGetStatus_1.Done = 1)THEN
    //--> Connection is OK, further actions can be done
    IF (UA_ConnectionGetStatus_1.ConnectionStatus = UACS_Connected) THEN
      (*-------- Condition for next step ------------------*)
      FB_do_Case := 20;  // --> further actions
      END_IF
    END_IF
  //--> Error Handling
  IF (UA_ConnectionGetStatus_1.Error = 1) THEN
    FB_do_Case := 99;  // --> Error handling
  END_IF
  //--> FB release --
  UA_ConnectionGetStatus_1(Execute := FALSE,
    ConnectionHdl := ConnectionHdl,
    Timeout := T#1s);
END_IF
```

Display of the connection status in the Automation Studio Watch window

If the connection is correct

<table>
<thead>
<tr>
<th>ConnectionGetStatus_1</th>
<th>UA_Conn</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>ConnectionHdl</td>
<td>DWORD</td>
<td>35588088</td>
</tr>
<tr>
<td>Timeout</td>
<td>TIME</td>
<td>T#0s</td>
</tr>
<tr>
<td>ConnectionStatus</td>
<td>UACone</td>
<td>UACS_Connected</td>
</tr>
<tr>
<td>ServerState</td>
<td>UServer</td>
<td>UASS_Running</td>
</tr>
<tr>
<td>ServiceLevel</td>
<td>BYTE</td>
<td>255</td>
</tr>
<tr>
<td>Done</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>Busy</td>
<td>BOOL</td>
<td>FALSE</td>
</tr>
<tr>
<td>Error</td>
<td>BOOL</td>
<td>FALSE</td>
</tr>
<tr>
<td>ErrorID</td>
<td>DWORD</td>
<td>0</td>
</tr>
</tbody>
</table>

During restart of the bus controller

<table>
<thead>
<tr>
<th>NodeReleaseHandle_1</th>
<th>UA_Node</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>ConnectionHdl</td>
<td>DWORD</td>
<td>81350320</td>
</tr>
<tr>
<td>Timeout</td>
<td>TIME</td>
<td>T#0s</td>
</tr>
<tr>
<td>ConnectionStatus</td>
<td>UACone</td>
<td>UACS_ConnectionError</td>
</tr>
<tr>
<td>ServerState</td>
<td>UServer</td>
<td>UASS_Unknown</td>
</tr>
<tr>
<td>ServiceLevel</td>
<td>BYTE</td>
<td>0</td>
</tr>
</tbody>
</table>
10.3.4 UA_GetNamespaceIndex()

This function block is used to read out the used NamespaceIndex of the OPC UA bus controller firmware.

- Different namespace indexes can be read according to the requested "NamespaceUri".

<table>
<thead>
<tr>
<th>NamespaceIndex</th>
<th>NamespaceUri</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;<a href="http://opcfoundation.org/UA/">http://opcfoundation.org/UA/</a>&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;urn:br-automation/BuR/UA/X20BC008U&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;<a href="http://br-automation.com/OpcUa/BC/io-system/">http://br-automation.com/OpcUa/BC/io-system/</a>&quot;</td>
</tr>
</tbody>
</table>

Information:

To avoid possible errors and problems in applications, the NamespaceIndex of "http://br-automation.com/OpcUa/BC/io-system/" should always be determined using this function block. See also footnote 2 in the table of "Namespaces used" on page 18.

- The specification of the relative path name is standardized according to the BNF format.

Example

Definition of the relative path to "DeviceSet / X20BC008U / X2X / SubDevices".

Direct definition

```
RelativePath_ObjectNode:= '/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices';
```

Indirect definition

```
NamespaceIndex_BC_Str := '2'; // String, identical with FB data from NamespaceIndex

RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode), ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode), ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode), ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode), ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode), ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode), ADR(':X2X/'));
strcat(ADR(RelativePath_ObjectNode), ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode), ADR(':SubDevices'));
```
10.3.5 UA_TranslatePath()

Objects (directory, data, methods, I/O data, etc.) can be uniquely referenced using the directory path (browse path) and object name, or if known, the NodeID.

This function block uses the specified directory or object data to determine the NodeID. This NodeID is the basic prerequisite for all further accesses to objects or methods.

<table>
<thead>
<tr>
<th>StartNodeID.Identifier from NamespaeIndex 0</th>
<th>Value from NamespaeIndex 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartNodeID.Identifier := '84';</td>
<td>&quot;Root&quot; from Namespacelndex 0</td>
</tr>
<tr>
<td>StartNodeID.Identifier := '85';</td>
<td>&quot;Objects&quot; from Namespacelndex 0</td>
</tr>
</tbody>
</table>

**Information:**

The node to be determined is defined via the StartNodeID and relative path name.

- Names and NodeIDs from NamespaceIndex 0 are defined by the standard and therefore unchangeable.

- The OPC UA bus controller supports only IdentifierType "UAIdentifierType_Numeric". The specification of the object name using a string (UAIdentifierType_String) is not supported.

- The specification of the relative path name is standardized according to the BNF format.

**Application tip**

The fewer sub-paths specified in the relative path when determining NodeIDs using UA_TranslatePath(), the faster the search process becomes. By selecting a favorable StartNodeID, determining the NodeIDs can be significantly accelerated. To do this, the NodeID of the start path must be determined once and then passed as StartNodeID for the subsequent TranslatePath operations.

See also the information about NamespaceIndex 2 in "Namespaces used" on page 18.

### 10.3.5.1 Determining the NodeID of an X2X module

In this example, the NodeID of X2X module "X20AT2402" should be determined in slot 3.

**Procedure**

1. StartNodeID is "Root / Objects": "/0:Root /0:Objects" = "85"
2. Definition of the relative path name "/2:DeviceSet /2:X20BC008U /2:X2X /2:SubDevices /2:ST3"
3. Execute UA_TranslatePath() as soon as Execute = TRUE.
4. Wait until Busy = FALSE.
5. After correct execution of the function block, specify the determined NodeID to the subsequent function or store it temporarily.
Application tip

As soon as several X2X modules should be read, it is advantageous to first determine the NodeID from path "DeviceSet / X20BC008U / X2X / SubDevices" and to specify it as StartNodeID for further TranslatePath calls. This makes the relative path name shorter and the search process faster.

Representation of the information model using UaExpert

Example program for UA_TranslatePath()

Initialization

Only call once when TranslatePath() should be started.

```c
(*----------------------------------------------------*)
// INIT UA_TranslatePath() – only once at INIT
(*----------------------------------------------------*)
// 1 .. UAIdentifierType_String       ! NOT supported for UA_TranslatePath() !
// 2 .. UAIdentifierType_Numeric
(*----------------------------------------------------*)
StartNodeID.NamespaceIndex := 0;
StartNodeID.Identifier := '85';                    // <Objects>
StartNodeID.IdentifierType := UAIdentifierType_Numeric;
IO_RelativePath_slot := 3;                       // Module Slot Nu.
itos(IO_RelativePath_slot,ADR(IO_Search_Slot));        // Prepare Search Slot String
// generate RelativePath
RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR('X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR('X2X/'));
strcat(ADR(RelativePath_ObjectNode),ADR('SubDevices/'));
strcat(ADR(RelativePath_ObjectNode),ADR('ST'));
strcat(ADR(RelativePath_ObjectNode),ADR(IO_Search_Slot)); // add prepared Search Slot String
ExecuteTranslatePath_0 := TRUE;                           // Enable FB
```
Cyclic program section

```c
//-- Cyclic program --> NodeID will be determined, as soon as ExecuteTranslatePath_0 = TRUE
UA_TranslatePath_0(Execute       := ExecuteTranslatePath_0,
                    ConnectionHdl := ConnectionHdl,
                    StartNodeID   := StartNodeID,
                    RelativePath  := RelativePath_ObjectNode,
                    Timeout       := T#5s);

IF (UA_TranslatePath_0.Busy = 0) THEN
  ExecuteTranslatePath_0 := FALSE;          //-> Release the FB for further use
  IF (UA_TranslatePath_0.Done = 1) THEN
    (*-------- Parameter for next function block --------*)
    UA_NodeGetHandle_0.NodeID := UA_TranslatePath_0.TargetNodeID;
    (*---------------------------------------------------*)
    ExecuteNodeGetHandle_0 := TRUE;       //-> Enable NodeGetHandle
  END_IF
ENDIF
```

10.3.5.2 Determining the NodeID of a method path and method

In this example, the NodeID of method `ApplyAllConfigurations` should be determined. However, since the NodeID of the method path may be required for further method calls in the application, it is determined beforehand and used as the StartNodeID for the second `UA_TranslatePath()` call.

**Procedure**

1. StartNodeID is "Root": "/0:Root" = "84"
2. Definition of the relative path name "/0:Objects /2:DeviceSet /2:X20BC008U /2:MethodSet".
3. Execute `UA_TranslatePath()` as soon as Execute = TRUE.
4. Wait until Busy = FALSE.
5. After correct execution of the function block, apply the determined NodeID as the StartNodeID.
6. Definition of the relative path name "/2:ApplyAllConfigurations".
7. Execute `UA_TranslatePath()` as soon as Execute = TRUE.
8. Wait until Busy = FALSE.
9. After correct execution of the function block, specify the determined NodeID to the subsequent function or store it temporarily.

**Sample program**

**Determining the NodeID for the method path**

```c
(* UA_TranslatePath - get ObjectNodeID of method path *)
StartNodeID_root.NamespaceIndex := 0;
StartNodeID_root.Identifier     := '84';
StartNodeID_root.IdentifierType := UAIdentifierType_Numeric;

// Path= /0:Objects/2:DeviceSet/2:X20BC008U/2:MethodSet'
RelativePath_ObjectNode := '/0:Objects/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':MethodSet'));

UA_TranslatePath_0(Execute       := ExecuteTranslatePath_0,
                    ConnectionHdl := ConnectionHdl,
                    StartNodeID   := StartNodeID_root,
                    RelativePath  := RelativePath_ObjectNode,
                    Timeout       := T#10s);

IF (UA_TranslatePath_0.Busy = 0) THEN
  ExecuteTranslatePath_0 := 0;
  IF (UA_TranslatePath_0.Done = 1) THEN
    ErrorID := 0;
    (*-------- Parameter for next function block --------*)
  END_IF
ENDIF
```
Determining the NodeID for method "ApplyAllConfigurations"

NodeID of the method path is used as the StartNodeID for the new call.

```c
ObjectNodeID := UA_TranslatePath_0.TargetNodeID;
(*-------------------------*)
END_IF

IF (UA_TranslatePath_0.Error = 1) THEN
  ErrorID := UA_TranslatePath_0.ErrorID;
END_IF
END_IF
```

```c
Determining the NodeID for method "ApplyAllConfigurations"

NodeID of the method path is used as the StartNodeID for the new call.

(* UA_TranslatePath - set StartNodeID = ID of method path = get MethodNodeID ApplyAll *)
StartNodeID_ObjectNode.NamespaceIndex := ObjectNodeID.NamespaceIndex;
StartNodeID_ObjectNode.Identifier     := ObjectNodeID.Identifier;
StartNodeID_ObjectNode.IdentifierType := ObjectNodeID.IdentifierType;

// Path= /2:ApplyAllConfigurations';
RelativePath_ApplyAll := '/';
strcat(ADR(RelativePath_ApplyAll),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ApplyAll),ADR(':ApplyAllConfigurations'));

UA_TranslatePath_1(Execute := ExecuteTranslatePath_1,
  ConnectionHdl := ConnectionHdl,
  StartNodeID   := StartNodeID_ObjectNode,
  RelativePath  := RelativePath_ApplyAll,
  Timeout       := T#10s);

IF (UA_TranslatePath_1.Busy = 0) THEN
  ExecuteTranslatePath_1 := 0;
  IF (UA_TranslatePath_1.Done = 1) THEN
    ErrorID := 0;
    (*-------- Parameter for next function block --------*)
    MethodNodeID_ApplyAll := UA_TranslatePath_1.TargetNodeID;
    (*-------------------------*)
  END_IF
END_IF

IF (UA_TranslatePath_1.Error = 1) THEN
  ErrorID := UA_TranslatePath_1.ErrorID;
END_IF
END_IF
```
10.3.6 UA_NodeGetHandle()

This function block is used to define and reserve a handle based on the specified NodeID or directory or object data.

Information:

- Each additional time this function block is executed, a new handle is reserved even if the requested NodeID remains the same.
- The handle "NodeHdl" returned by the function block is retained during the entire session. Any number of read and write accesses can be performed.
- Obsolete or no longer required handles must be freed up again with UA_NodeReleaseHandle().

Sample program

Initialization

Only call once before each enabling of NodeGetHandle.

```plaintext
// NodeID was determined using UA_TranslatePath() and transferred
ObjectNodeID_Di := UA_TranslatePath_0.TargetNodeID;
ExecuteNodeGetHandle_0 := TRUE; //-> Enable NodeGetHandle
```

Cyclic program section

```plaintext
/* UA_NodeGetHandle - get a handle for required DigitalInput node */
UA_NodeGetHandle_0(Execute := ExecuteNodeGetHandle_0,
ConnectionHdl := ConnectionHdl,
NodeID := ObjectNodeID_Di,
Timeout := T#10s);

IF (UA_NodeGetHandle_0.Busy = 0) THEN
  ExecuteNodeGetHandle_0 := 0;
  IF (UA_NodeGetHandle_0.Done = 1) THEN
    ErrorID := 0;
    (-- Parameter for next function block ---
    NodeHdl_Di := UA_NodeGetHandle_0.NodeHdl;
    (-- End parameter for next function block ---)
  END_IF
  IF (UA_NodeGetHandle_0.Error = 1) THEN
    ErrorID := UA_NodeGetHandle_0.ErrorID;
    NodeHdl_Di := 0;
  END_IF
END_IF
```

10.3.7 UA_NodeReleaseHandle()

This function block is used to free up a handle reserved by UA_NodeGetHandle().

- Obsolete or no longer necessary handles must be freed up again with this function block.

Sample program

Cyclic program section

```c
UA_NodeReleaseHandle_0(Execute := ExecuteNodeReleaseHandle_0,
    ConnectionHdl := ConnectionHdl,
    NodeHdl := NodeHdl_Di,
    Timeout := T#10s);

IF (UA_NodeReleaseHandle_0.Busy = 0) THEN
    ExecuteNodeReleaseHandle_0 := 0;
    IF (UA_NodeReleaseHandle_0.Done = 1) THEN
        ErrorID := 0;
        NodeHdl_Di := 0;
        END_IF
    IF (UA_NodeReleaseHandle_0.Error = 1) THEN
        ErrorID := UA_NodeReleaseHandle_0.ErrorID;
    END_IF
END_IF
```
Programming example

10.3.8 UA_Read()

This function block is used to read data from objects.

```
(* INITIAL SETUP before UA_Read() *)
Variable_Rd0 := '::OpcTest:BC_SoftwareRevision';
BC_SoftwareRevision := 'V?.??'; // init Target Variable for UA_Read()
```

10.3.8.1 Reading a known NodeID

In this example, the software revision number of the bus controller should be read out. The read process is performed using the NodeID (NamespaceIndex = 2, Identifier = 305) known from UaExpert.

Information:
Using hard-coded node IDs can cause problems since they may change with a different firmware version or after changes to the X2X Link network.

Advantages of direct access

Direct access to known NodeIDs can be realized faster from a programming point of view since the intermediate steps for determining the NodeIDs are omitted (e.g. determining an advantageous StartNode, taking into account NamespaceIndex and X2X slot, constructing the relative path name).

Disadvantages of direct access

This procedure is more error-prone, and NodeIDs can change during the development phase of projects or during firmware updates.

Representation of the information model using UaExpert
Declaring target variables in Automation Studio

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable_Rd0</td>
<td>STRING[255]</td>
</tr>
<tr>
<td>BC_SoftwareRevision</td>
<td>STRING[20]</td>
</tr>
<tr>
<td>TargetNodeID</td>
<td>UANodeID</td>
</tr>
<tr>
<td>TargetNodeHdl</td>
<td>DWORD</td>
</tr>
<tr>
<td>NodeAddHdl</td>
<td>UANodeAdditionalInfo</td>
</tr>
</tbody>
</table>

Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.
   
   ```
   UA_Connect_0()
   ```

2) Reserve the NodeID.
   Only 1 handle must be reserved for a known NodeID.
   
   ```
   UA_NodeGetHandle_0() // TargetNodeHdl := UA_NodeGetHandle_0.NodeHdl;
   ```

3) Execute the read command.
   
   ```
   UA_Read_0() // Variable_Rd0 := '::OpcTest:BC_SoftwareRevision;
   ```

4) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.
   
   ```
   UA_NodeReleaseHandle_0()
   ```

5) Close the connection, see example "UA_Disconnect()" on page 73.
   
   ```
   UA_Disconnect_0()
   ```

Sample program

Initialization

Only call once.

```
(* INITIAL SETUP before UA_Read() *)
Variable_Rd0 := '::OpcTest:BC_SoftwareRevision';
BC_SoftwareRevision := 'V?.??'; // Variable for UA_Read()

(* define TargetNodeID *)
TargetNodeID.NamespaceIndex := 2; // Namespaceindex = 2
TargetNodeID-Identifier := '305'; // hardcoded NodeID-Identifier
TargetNodeID-IdentifierType := UAIentifierType_Numeric;
```
Cyclic program section

(* UA_NodeGetHandle - get a handle for required node *)
UA_NodeGetHandle_0(Execute := ExecuteNodeGetHandle_0,
ConnectionHdl := ConnectionHdl,
NodeID := TargetNodeID,
Timeout := T#10s);

IF (UA_NodeGetHandle_0.Busy = 0) THEN
  ExecuteNodeGetHandle_0 := 0;
  IF (UA_NodeGetHandle_0.Done = 1) THEN
    (*-------- Parameter for next function block --------*)
    TargetNodeHdl := UA_NodeGetHandle_0.NodeHdl;
    (*---------------------------------------------------*)
  ELSIF (UA_NodeGetHandle_0.Error = 1) THEN
    TargetNodeHdl := 0;
  END_IF
END_IF

(* UA_Read - read required node from OPC-UA Server and write it in local plc variable *)
NodeAddInfo.AttributeId := UAAI_Value;  // read node value
NodeAddInfo.IndexRangeCount := 0;

UA_Read_0(Execute := ExecuteRead_0,
  ConnectionHdl := ConnectionHdl,
  NodeHdl := TargetNodeHdl,
  NodeAddInfo := NodeAddInfo,
  Variable := Variable_Rd0,
  Timeout := T#5s);

IF (UA_Read_0.Busy = 0) THEN
  ExecuteRead_0 := FALSE;  //quit Enable
  IF (UA_Read_0.Done = 1) THEN
    ErrorID := 0;
    (* --> Error Handling *)
  ELSIF (UA_Read_0.Error = 1) THEN
    ErrorID := UA_Read_0.ErrorID;
  END_IF
END_IF
10.3.8.2 Reading out a digital input

In this example, digital input "DigitalInput01" of the module in slot 2 should be read.

Representation of the information model using UaExpert

If the necessary data type for reading out is unknown, the target data type can be determined using UaExpert. In this example, a variable of data type "BOOL" is required.

Declaring target variables in Automation Studio

Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.

   UA_Connect_0() // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.


3) Determine the StartNodeID.
   StartNodeID.Identifier is "Root": "0:Root" = "84"

4) Determine the relative path name.

   // Path=/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices
   // 2:ST001/2:X05P59400
   // 2:ST002/2:X05D1372
   // 2:ST001/2:X05P59400
   // 2:ST002/2:X05D1372
   UA_TranslatePath_0() // ObjectNodeID_Di := UA_TranslatePath_0.TargetNodeID;

5) Reserve the handle.

   UA_NodeGetHandle_0() // NodeHdl_Di := UA_NodeGetHandle_0.NodeHdl;

6) Execute the read command.

   UA_Read_0() // Variable_Rd0 := '::Taskname:VarA_DigInput01';

7) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.

   UA_NodeReleaseHandle_0()
8) Close the connection, see example "UA_Disconnect()" on page 73.
   
   UA_Disconnect_0()

Sample program

(* UA_TranslatePath - get ObjectNodeID - DigitalInput *)
StartNodeID_Root.NamespaceIndex := 0;
StartNodeID_Root.Identifier     := '84';
StartNodeID_Root.IdentifierType := UAIdentifierType_Numeric;

// Path=/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST2/2:ParameterSet/2:DigitalInput01';
RelativePath_ObjectNode := '/0:Objects/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X2X/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':SubDevices/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ST2/')); // 'ST' and Slot of DigitalInput Module
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ParameterSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DigitalInput01'));

UA_TranslatePath_0(Execute       := ExecuteTranslatePath_0,
                   ConnectionHdl := ConnectionHdl,
                   StartNodeID   := StartNodeID_Root,
                   RelativePath  := RelativePath_ObjectNode,
                   Timeout       := T#10s);

IF (UA_TranslatePath_0.Busy = 0) THEN
  ExecuteTranslatePath_0 := 0;
  IF (UA_TranslatePath_0.Done = 1) THEN
    ErrorID         := 0;
    ObjectNodeID_Di := UA_TranslatePath_0.TargetNodeID;
    (*-------- Parameter for next function block --------*)
    (*---------------------------------------------------*)
  END_IF
ENDIF

IF (UA_TranslatePath_0.Error = 1) THEN
  ErrorID := UA_TranslatePath_0.ErrorID;
ENDIF

(* UA_NodeGetHandle - get a handle for required DigitalInput node *)
UA_NodeGetHandle_0(Execute       := ExecuteNodeGetHandle_0,
                   ConnectionHdl := ConnectionHdl,
                   NodeID        := ObjectNodeID_Di,
                   Timeout       := T#10s);

IF (UA_NodeGetHandle_0.Busy = 0) THEN
  ExecuteNodeGetHandle_0 := 0;
  IF (UA_NodeGetHandle_0.Done = 1) THEN
    ErrorID         := 0;
    NodeHdl_Di     := UA_NodeGetHandle_0.NodeHdl;
    (*-------- Parameter for next function block --------*)
    (*---------------------------------------------------*)
  END_IF
ENDIF

IF (UA_NodeGetHandle_0.Error = 1) THEN
  ErrorID := UA_NodeGetHandle_0.ErrorID;
  NodeHdl_Di := 0;
ENDIF

(* UA_Read - read required node from OPC-UA Server and write it in local plc variable *)
10.3.8.3 Reading information from data type "LocalizedText"

In this example, the name of the module should be read from slot 3.

Representation of the information model using UaExpert

Declaring the target variable in Automation Studio
Programming example

Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.
   UA_Connect_0() // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.

3) Determine the StartNodeID.
   StartNodeID.Identifier is "Root / Objects": "0:Root /0:Objects" = "85"

4) Determine the relative path name.
   // Path=/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST3/2:Model';
   UA_TranslatePath_0() // TargetNodeID := UA_TranslatePath_0.TargetNodeID;

5) Reserve the handle.
   UA_NodeGetHandle_0() // TargetNodeHdl := UA_NodeGetHandle_0.NodeHdl;

6) Execute the read command.
   UA_Read_0() // UA_READ_Model := '::OpcTest:X2X_Model';

7) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.
   UA_NodeReleaseHandle_0()

8) Close the connection, see example "UA_Disconnect()" on page 73.
   UA_Disconnect_0()

Sample program

Initialization

Only call once before TranslatePath() and the start of the read process.

(* INITIAL SETUP before UA_Read() *)
UA_READ_Model := '::OpcTest:X2X_Model';
X2X_Model.Text := '???'; // Variable for UA_Read()

(* INITIAL SETUP for UA_TranslatePath() *)
StartNodeID.NamespaceIndex := 0;
StartNodeID.Identifier := '85'; // <Objects>
StartNodeID.IdentifierType := UAIdentifierType_Numeric;
IO_Slot := 3; // Module Slot Number
itoa(IO_Slot,ADR(IO_Slot_String)); // Prepare Slot String

// Generate relative path
RelativePath_ObjectNode := '/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X2X/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':SubDevices/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ST3'));
strcat(ADR(RelativePath_ObjectNode),ADR(IO_Search_Slot)); // add prepared Search Slot String
strcat(ADR(RelativePath_ObjectNode),ADR('/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':Model'));

(* START execution of UA_TranslatePath() *)
(*-------- Parameter for next function block --------*)
ExecuteTranslatePath_0 := TRUE;
(*-----------------------------------------------*)
Cyclic program section

(* UA_TranslatePath - get Nodeid for required nodes *)
UA_TranslatePath_0(Execute       := ExecuteTranslatePath_0,
    ConnectionHdl := ConnectionHdl,
    StartNodeID   := StartNodeID,
    RelativePath  := RelativePath_ObjectNode,
    Timeout       := T#5s);

IF (UA_TranslatePath_0.Busy = 0) THEN
    ExecuteTranslatePath_0 := FALSE;    //-> Release the FB for further use
    (* start execution after setup parameter for UA_NodeGetHandle *)
    IF (UA_TranslatePath_0.Done = 1) THEN
        (*-------- Parameter for next function block --------*)
        TargetNodeID         := UA_TranslatePath_0.TargetNodeID;
        ExecuteNodeGetHandle_0 := TRUE;    //-> Enable NodeGetHandle
        (*---------------------------------------------------*)
        END_IF
    END_IF
END_IF

(* UA_NodeGetHandle - get a handle for required node *)
UA_NodeGetHandle_0(Execute       := ExecuteNodeGetHandle_0,
    ConnectionHdl := ConnectionHdl,
    NodeID        := TargetNodeID,
    Timeout       := T#10s);

IF (UA_NodeGetHandle_0.Busy = 0) THEN
    ExecuteNodeGetHandle_0 := 0;
    IF (UA_NodeGetHandle_0.Done = 1) THEN
        UA_NodeGetHandle_ok := TRUE;
        (*-------- Parameter for next function block --------*)
        TargetNodeHdl  := UA_NodeGetHandle_0.NodeHdl;
        ExecuteRead_0  := TRUE;       //-> Enable read
        (*---------------------------------------------------*)
        ELSIF (UA_NodeGetHandle_0.Error = 1) THEN
            UA_NodeGetHandle_ok := FALSE;
        END_IF
    END_IF
END_IF

(*----------------------------------------------------*)
(* UA_Read - read required node from OPC-UA Server
and write it in local plc variable *)
(*----------------------------------------------------*)
NodeAddInfo.AttributeId     := UAAI_Value;  // read node value
NodeAddInfo.IndexRangeCount := 0;
UA_READ_Model    := '::OpcTest:X2X_Model';
UA_Read_0(Execute       := ExecuteRead_0,
    ConnectionHdl := ConnectionHdl,
    NodeHdl       := TargetNodeHdl,
    NodeAddInfo   := NodeAddInfo,
    Variable      := UA_READ_Model,
    Timeout       := T#5s);

IF (UA_Read_0.Busy = 0) THEN
    ExecuteRead_0 := FALSE;      //quit Enable
    (* --> Prepare next Step – check READ DATA *)
    IF (UA_Read_0.Done = 1) THEN
        ErrorID := 0;
        (* --> Error Handling *)
        ELSIF (UA_Read_0.Error = 1) THEN
            ErrorID := UA_Read_0.ErrorID;
        END_IF
    END_IF
END_IF
### 10.3.9 UA_Write()

This function block is used to write data to objects.

#### Information:

For read or write access, matching process variables must be created in the data type in the Automation Studio program. For more information, see "Node identification" on page 31.

```plaintext
(* INITIAL SETUP before UA_Write() *)
Variable_Wr0 := '::OpcTest:DigitalOutput';
DigitalOutput := FALSE;  // init Write Variable for UA_Write()
```

#### 10.3.9.1 Writing to a digital output

In this example, digital output "DigitalOutput01" of the module in slot 3 should be written.

**Representation of the information model using UaExpert**

If the necessary data type for reading out is unknown, the target data type can be determined using UaExpert. In this example, a variable of data type "BOOL" is required.

**Declaring target variables in Automation Studio**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NamespaceIndex</td>
<td>2</td>
</tr>
<tr>
<td>IdentifierType</td>
<td>Numeric</td>
</tr>
<tr>
<td>Identifier</td>
<td>212992</td>
</tr>
<tr>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>BrowseName</td>
<td>&quot;DigitalOutput01&quot;</td>
</tr>
<tr>
<td>DisplayName</td>
<td>&quot;DigitalOutput01&quot;</td>
</tr>
<tr>
<td>Description</td>
<td>&quot;24 VDC / 0.5 A&quot;</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>06.01.2017 05:01:46:475</td>
</tr>
<tr>
<td>SourceTime</td>
<td>06.01.2017 05:01:46:475</td>
</tr>
<tr>
<td>SourcePicoSeconds</td>
<td>0</td>
</tr>
<tr>
<td>SourceTimestamp</td>
<td>06.01.2017 05:01:46:475</td>
</tr>
<tr>
<td>SourcePicoSeconds</td>
<td>0</td>
</tr>
<tr>
<td>SourcePicoSeconds</td>
<td>0</td>
</tr>
<tr>
<td>SourcePicoSeconds</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>true</td>
</tr>
<tr>
<td>DataType</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

```plaintext
StartNodeId = UANode; RelativePath = ObjectNode; ObjectId = UANode; NodeHdl = DWORD; NodeAddInfo = UANodeAdditionalInfo; VarA_DigitalOut01 = BOOL; TimestampVarA = DATE_AND_TIME; VariableWr0 = STRING[255]; UA_Write_0 = UA_Write;
```
Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.
   
   ```
   UA_Connect_0() // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'
   ```

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.
   
   ```
   UA_GetNamespaceIndex_BC() // NamespaceUri := 'http://br-automation.com/OpcUa/BC/io-system/'
   ```

3) Determine the StartNodeID.
   StartNodeID.Identifier is "Root": "0:Root" = "84"

4) Determine the relative path name.
   
   ```
   // Path=/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices
   //      /2:ST3/2:ParameterSet/2:DigitalOutput01';
   UA_TranslatePath_0() // ObjectNodeID := UA_TranslatePath_0.TargetNodeID;
   ```

5) Reserve the handle.
   
   ```
   UA_NodeGetHandle_0() // NodeHdl_Di := UA_NodeGetHandle_0.NodeHdl;
   ```

6) Execute the write command.
   
   ```
   UA_Write_0() // Variable_Wr0 := '::Taskname:VarA_DigOutput01';
   ```

7) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.
   
   ```
   UA_NodeReleaseHandle_0()
   ```

8) Close the connection, see example "UA_Disconnect()" on page 73.
   
   ```
   UA_Disconnect_0()
   ```

Sample program

```
(* UA_TranslatePath - get ObjectNodeID - DigitalOutput *)
StartNodeID_Root.NamespaceIndex := 0;
StartNodeID_Root.Identifier := '84';
StartNodeID_Root.IdentifierType := UAIdentifierType_Numeric;

// Path=/0:Objects/2:DeviceSet/2:X20BC008U/2:X2X/2:SubDevices/2:ST2/2:ParameterSet/2:DigitalOutput01';
RelativePath_ObjectNode := '/0:Objects/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X2X/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':SubDevices/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ST2/')); // 'ST' and Slot of DigitalOutput Module
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ParameterSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DigitalOutput01'));

UA_TranslatePath_0(Execute       := ExecuteTranslatePath_0,
   ConnectionHdl := ConnectionHdl,
   StartNodeID   := StartNodeID_Root,
   RelativePath  := RelativePath_ObjectNode,
   Timeout       := T#10s);

IF (UA_TranslatePath_0.Busy = 0) THEN
   ExecuteTranslatePath_0 := 0;

   IF (UA_TranslatePath_0.Done = 1) THEN
     ErrorID := 0;
     (*-------- Parameter for next function block --------*)
     ObjectNodeID_Do := UA_TranslatePath_0.TargetNodeID;
     (*---------------------------------------------------*)
   END_IF

IF (UA_TranslatePath_0.Error = 1) THEN
   ErrorID := UA_TranslatePath_0.ErrorID;
END_IF
```
Programming example

END_IF

(* UA_NodeGetHandle - get a handle for required DigitalOutput node *)
UA_NodeGetHandle_0(Execute := ExecuteNodeGetHandle_0,
                       ConnectionHdl := ConnectionHdl,
                       NodeID := ObjectNodeID_Do,
                       Timeout := T#10s);

IF (UA_NodeGetHandle_0.Busy = 0) THEN
  ExecuteNodeGetHandle_0 := 0;
  IF (UA_NodeGetHandle_0.Done = 1) THEN
    ErrorID := 0;
    (*-------- Parameter for next function block --------*)
    NodeHdl_Do := UA_NodeGetHandle_0.NodeHdl;
    (*---------------------------------------------------*)
  END_IF
  IF (UA_NodeGetHandle_0.Error = 1) THEN
    ErrorID := UA_NodeGetHandle_0.ErrorID;
    NodeHdl_Do := 0;
  END_IF
END_IF

(* UA_Write - read required node from OPC-UA Server and write it in local plc variable *)
NodeAddInfo_0.AttributeId := UAAI_Value;
NodeAddInfo_0.IndexRangeCount := 0;
Variable_Wr0 := '::Taskname:VarA_DigOutput01';

UA_Write_0(Execute := ExecuteWrite_0,
                       ConnectionHdl := ConnectionHdl,
                       NodeHdl := NodeHdl_Di,
                       NodeAddInfo := NodeAddInfo_0,
                       Timeout := T#10s,
                       Variable := Variable_Wr0);

IF (UA_Write_0.Busy = 0) THEN
  ExecuteWrite_0 := 0;
  IF (UA_Write_0.Done = 1) THEN
    ErrorID := 0;
    TimestampVarA := UA_Write_0.TimeStamp;
  END_IF
  IF (UA_Write_0.Error = 1) THEN
    ErrorID := UA_Write_0.ErrorID;
  END_IF
END_IF
10.3.10 UA_NodeGetHandleList()

This function block is used to define and reserve handles based on the passed list of NodeID or directory or object data.

- The handles returned by the function block in array "NodeHdls" are retained during the entire session. Any number of read and write accesses can be performed.
- For each handle requested, there is an error or OK entry in the error list that is also returned. If an error occurred while reserving a handle, an error message is entered on the corresponding list offset of the NodeErrorId list.
- These handles remain assigned until they are freed up using function block "UA_NodeReleaseHandleList()".

Sample program

Initialization

Only call once before each enabling of NodeGetHandle.

```plaintext
(*----------------------------------------------------*)
// Parameters for UA_NodeGetHandleList() - max. 64 entries
(*----------------------------------------------------*)
UA_NodeHdlList_Count := 2; // Number of list entries

UA_NodeID_List[0].NamespaceIndex := NamespaceIndex;
// '14' ConfigurationStatusAll (Int32)
UA_NodeID_List[0].Identifier := NodeID_ConfigStatusAll;
UA_NodeID_List[0].IdentifierType := UAIdentifierType_Numeric;

UA_NodeID_List[1].NamespaceIndex := NamespaceIndex;
// '15' ConfigurationStatusArray (Int32 Array 0..200)
UA_NodeID_List[1].Identifier := NodeID_ConfStatArray;
UA_NodeID_List[1].IdentifierType := UAIdentifierType_Numeric;

Cyclic program section

(*----------------------------------------------------*)
// UA_NodeGetHandleList() - max. 64 entries
// UA_NodeReleaseHandleList()
(*----------------------------------------------------*)

//-- Reserve/free handle --
UA_NodeGetHandleList_0(Execute := ExecuteNodeGetHandleList_0,
NodeIDCount := UA_NodeHdlList_Count, // 3
NodeIDs := UA_NodeID_List,
ConnectionHdl := ConnectionHdl,
Timeout := T#10s);

IF (UA_NodeGetHandleList_0.Busy = 0) THEN
  ExecuteNodeGetHandleList_0 := 0;  //-> Release the FB for further use

  IF (UA_NodeGetHandleList_0.Done = 1) THEN
    IF (UA_NodeGetHandleList_0.Error = 1) THEN
      UA_NodeGetHandleList_ok := FALSE;
    END_IF
  END_IF
END_IF
```
10.3.11 UA_ReadList()

This function block is used to read several nodes simultaneously. The desired nodes and target variables in the Automation Studio program are defined in corresponding lists beforehand.

Information:

For read or write access, matching process variables must be created in the data type in the Automation Studio program. For more information, see "Node identification" on page 31.

(* INITIAL SETUP before UA_ReadList() *)
Variable_RdList_0[0] := '::OpcTest:UAc_ConfigurationStatusAll';
UAc_ConfigurationStatusAll := 0;  // init Target Variable for UA_Read()

10.3.11.1 Reading a list

In this example, the following objects should be read in a single reading process:

- The entire configuration status
- The status array with the collection of all ModuleOk objects
- The status array with the collection of all ConfigurationStatus objects of the individual X2X modules

Information:

Status arrays can only be read. Write accesses are not possible.

Representation of the information model using UaExpert
Declaring target variables and arrays for the read data in Automation Studio

When declaring the target variables, the associated program module must also be specified.

**Example**

<table>
<thead>
<tr>
<th>Program module</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>::OpcTest:UAc_ConfigurationStatusAll</td>
<td>Variable_RdList_0[0] := '::OpcTest:UAc_ConfigurationStatusAll'; // StatusAll</td>
</tr>
<tr>
<td>::OpcTest:UAc_ConfigurationStatusArray</td>
<td>Variable_RdList_0[1] := '::OpcTest:UAc_ConfigurationStatusArray'; // Array</td>
</tr>
<tr>
<td>::OpcTest:UAc_ModuleOkArray</td>
<td>Variable_RdList_0[2] := '::OpcTest:UAc_ModuleOkArray'; // Array</td>
</tr>
</tbody>
</table>

- The length of these arrays is set to a fixed length of 201 entries in bus controller firmware versions > V1.06. With firmware versions < 1.06, the length of these arrays depends on the number of modules connected plus some reserve slots. The necessary array length must be determined with UaExpert since a length query via function block is not supported.

- The length of all arrays configured in Automation Studio must exactly match the bus controller arrays.

**Information:**

- Longer or shorter arrays cause an access error message from the function blocks in library AsOpcUac.
- The offset of these arrays corresponds to the slot number (e.g. UAc_ModuleOkArray[2] for slot 2).

**Procedure**

1) Establish a connection, see example "UA_Connect()" on page 71.

   ```
   UA_Connect_0()               // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'
   ```

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.

   ```
   UA_GetNamespaceIndex_BC()    // NamespaceUri := 'http://br-automation.com/OpcUa/BC/io-system/'
   // NamespaceIndex := 2; ... von Funktionsbausteinen ermittelte Index
   ```

3) Determine the StartNodeID.

   ```
   StartNodeID.Identifier is "Root": "0:Root" = "84"
   ```

4) Determine the relative path name.

   Since several paths must be determined, first select suitable StartNodeIDs.

   ```
   // Path="/0:Objects/2:DeviceSet/2:X20BC008U/2:ParameterSet"
   ```

   Accept the determined NodeID as StartNodeID and determine further path names.

   ```
   //Path_StatusAll := '/2:ConfigurationStatusAll'
   //Path_StatusArray := '/2:ConfigurationStatusArray'
   //Path_ModuleOK := '/2:ModuleOkArray'
   ```

5) Reserve handles.

   ```
   UA_NodeGetHandleList_0()    
   ```

6) Execute a read command for lists.

   ```
   UA_ReadList_0()            
   ```

7) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.

   ```
   UA_NodeReleaseHandle_0()   
   ```

8) Close the connection, see example "UA_Disconnect()" on page 73.

   ```
   UA_Disconnect_0()          
   ```
Sample program

Initialization

Only call once.

- Determine a reasonable StartNodeID using UA_TranslatePath() for accelerated readout of the actual NodeIDs. The received NodeID is specified as the StartNodeID for further searches. In this example, all required objects come from the same sub-path "ParameterSet".

- To avoid possible problems with a later firmware update, all NodeIDs should be determined using UA_TranslatePath().

```plaintext
(* UA_TranslatePath - get ObjectNodeID of Parameter path *)
StartNodeID_Root.NamespaceIndex := 0;
StartNodeID_Root.Identifier := '84';
StartNodeID_Root.IdentifierType := UAIdentifierType_Numeric;

// Path= /0:Objects/2:DeviceSet/2:X20BC008U/2:ParameterSet';
RelativePath_ObjectNode := '/0:Objects/';
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':DeviceSet/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':X20BC008U/'));
strcat(ADR(RelativePath_ObjectNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_ObjectNode),ADR(':ParameterSet'));

UA_TranslatePath_0(Execute := Execute.TranslatePath_0,
ConnectionHdl := ConnectionHdl,
StartNodeID := StartNodeID_Root,
RelativePath := RelativePath_ObjectNode,
Timeout := T#10s);

IF (UA_TranslatePath_0.Busy = 0) THEN
  Execute.TranslatePath_0 := 0;
  IF (UA_TranslatePath_0.Done = 1) THEN
    ErrorID := 0;
    ObjectNodeID := UA_TranslatePath_0.TargetNodeID;
    END_IF
  END_IF
ELSE
  ErrorID := UA_TranslatePath_0.ErrorID;
END_IF
END_IF

Using the NodeID from path "ParameterSet" as StartNodeID for further UA_TranslatePath() operations

(* UA_TranslatePath - set StartNodeID to ID of Parameter path *)
StartNodeID_ObjectNode.NamespaceIndex := ObjectNodeID.NamespaceIndex;
StartNodeID_ObjectNode.Identifier := ObjectNodeID.Identifier;
StartNodeID_ObjectNode.IdentifierType := ObjectNodeID.IdentifierType;

// Path= /2:ConfigurationStatusAll';
RelativePath_TargetNode := '/';
strcat(ADR(RelativePath_TargetNode),ADR(NamespaceIndex_BC_Str));
strcat(ADR(RelativePath_TargetNode),ADR(':ConfigurationStatusAll'));

UA_TranslatePath_1(Execute := Execute.TranslatePath_1,
ConnectionHdl := ConnectionHdl,
StartNodeID := StartNodeID_ObjectNode,
RelativePath := RelativePath_ApplyAll,
Timeout := T#10s);

IF (UA_TranslatePath_1.Busy = 0) THEN
  Execute.TranslatePath_1 := 0;
  IF (UA_TranslatePath_1.Done = 1) THEN
    ErrorID := 0;
    NodeID_ConfigStatAll := UA_TranslatePath_1.TargetNodeID;
    END_IF
  END_IF
ELSE
  ErrorID := UA_TranslatePath_1.ErrorID;
END_IF
END_IF
```
IF (UA_TranslatePath_1.Error = 1) THEN
  ErrorID := UA_TranslatePath_1.ErrorID;
END_IF

END_IF

- Determine all necessary NodeIDs using function block UA_TranslatePath().

<table>
<thead>
<tr>
<th>UaExpert name</th>
<th>Relative path</th>
<th>NodeID buffer</th>
<th>Target process variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationStatusAll</td>
<td>&quot;2.ConfigurationStatusAll&quot;</td>
<td>NodeID_ConfigStatAll</td>
<td>UAc_ConfigurationStatusAll</td>
</tr>
<tr>
<td>ConfigurationStatusArray</td>
<td>&quot;2.ConfigurationStatusArray&quot;</td>
<td>NodeID_ConfStatArray</td>
<td>UAc_ConfigurationStatusArray</td>
</tr>
<tr>
<td>ModuleOkArray</td>
<td>&quot;2.ModuleOkArray&quot;</td>
<td>NodeID_ModOkArray</td>
<td>UAc_ModuleOkArray</td>
</tr>
</tbody>
</table>

Specifying parameters to function blocks "UA_NodeGetHandleList()" and "UA_ReadList()"

```plaintext
(*----------------------------------------------------*)
// Parameters for UA_NodeGetHandleList() - max. 64 entries
(*----------------------------------------------------*)
UA_NodeHdlList_Count := 3;  // Number of list entries

NodeAddInfoList_0[0].AttributeId := UAAI_Value;
NodeAddInfoList_0[0].IndexRangeCount := 0;

NodeAddInfoList_0[1].AttributeId := UAAI_Value;
NodeAddInfoList_0[1].IndexRangeCount := 1;
NodeAddInfoList_0[1].IndexRange[0].StartIndex := 0;
NodeAddInfoList_0[1].IndexRange[0].EndIndex := 200;
  // ConfigurationStatusArray

NodeAddInfoList_0[2].AttributeId := UAAI_Value;
NodeAddInfoList_0[2].IndexRangeCount := 1;
NodeAddInfoList_0[2].IndexRange[0].StartIndex := 0;
NodeAddInfoList_0[2].IndexRange[0].EndIndex := 200;
  // ModuleOkArray

UA_NodeID_List[0].NamespaceIndex := NamespaceIndex;
UA_NodeID_List[0].Identifier := NodeID_ConfigStatusAll;
  // '14' ConfigurationStatusAll (Int32)
UA_NodeID_List[0].IdentifierType := UAIdentifierType_Numeric;

UA_NodeID_List[1].NamespaceIndex := NamespaceIndex;
UA_NodeID_List[1].Identifier := NodeID_ConfStatArray;
  // '15' ConfigurationStatusArray (Int32 Array 0..200)
UA_NodeID_List[1].IdentifierType := UAIdentifierType_Numeric;

UA_NodeID_List[2].NamespaceIndex := NamespaceIndex;
UA_NodeID_List[2].Identifier := NodeID_ModOkArray;
  // '16' ModuleOkArray (Boolean Array 0..200)
UA_NodeID_List[2].IdentifierType := UAIdentifierType_Numeric;

// Ua_ReadList() die in HandleList konfigurierten Daten auslesen
Variable_RdList_0[0] := ':OpcTest:UAc_ConfigurationStatusAll';  // StatusAll
Variable_RdList_0[1] := ':OpcTest:UAc_ConfigurationStatusArray';  // Array
Variable_RdList_0[2] := ':OpcTest:UAc_ModuleOkArray';  // Array
```
Calling function block UA_NodeGetHandleList()

- After the function block has been enabled with "ExecuteNodeGetHandleList_0:= TRUE", the required handles for the subsequent UA_ReadList accesses are determined and returned in "UA_NodeGetHandleList_0.NodeHdls".
- There is also an error or OK entry in an error list for each handle requested.

```plaintext
//-- Reserve/free handle --
UA_NodeGetHandleList_0(Execute := ExecuteNodeGetHandleList_0,
NodeIDCount := UA_NodeHdlList_Count, // 3
NodeIDs := UA_NodeID_List,
ConnectionHdl := ConnectionHdl,
Timeout := T#10s);

IF (UA_NodeGetHandleList_0.Busy = 0) THEN
  ExecuteNodeGetHandleList_0 := 0; //-> Release the FB for further use
IF (UA_NodeGetHandleList_0.Done = 1) THEN
  UA_NodeGetHandleList_ok := TRUE;
ELSIF (UA_NodeGetHandleList_0.Error = 1) THEN
  UA_NodeGetHandleList_ok := FALSE;
END_IF
END_IF
```

Cyclic program section for UA_ReadList()

- After the function block has been enabled with "ExecuteReadList_0:= TRUE", the handles specified in the "UA_NodeGetHandleList_0.NodeHdls" handle list are read out using UA_ReadList().
- During this read process, the data is only copied to the variables defined in list "Variable_Do_RdList_0" if a valid read access could be performed. This means that the handles were created correctly, node identification and data types are correct and the target variables were used at least once in the program initialization subroutine.

```plaintext
// Ua_ReadList() define HandleList read variables
Variable_RdList_0[0] := '::OpcTest:UAc_ConfigurationStatusAll'; // StatusAll
Variable_RdList_0[1] := '::OpcTest:UAc_ConfigurationStatusArray'; // Array
Variable_RdList_0[2] := '::OpcTest:UAc_ModuleOkArray'; // Array

Alternatively, only a subset of the configured handles can be read.

```plaintext
UA_ReadList_0(Execute := ExecuteReadList_0,
ConnectionHdl := ConnectionHdl,
NodeHdlCount := UA_NodeHdlList_Count, // 3
(*-------- Parameter for next function block --------*)
NodeHdls := UA_NodeGetHandleList_0.NodeHdls,
(*---------------------------------------------------*)
NodeAddInfo := NodeAddInfoList_0,
Variables := Variable_Do_RdList_0,
Timeout := T#5s);

IF (UA_ReadList_0.Busy = 0) THEN
  ExecuteReadList_0 := 0; //-> Release the FB for further use
END_IF
```
Evaluating the read data

After the list has been read, the application can search the ModuleOkArray to determine whether one of the modules does not have status "OK". The offset in the arrays corresponds to the slot.

In addition, the read data can be displayed in the Automation Studio Watch window.

10.4 Applications

10.4.1 Restarting the bus controller

In this use case, the bus controller is restarted using method Reboot.

Notes

• For information about method "Reboot", see "Reboot" on page 61.
• For information about calling methods, see "Executing methods using library "AsOpcUac"" on page 54.
• Fixed NodeIDs

The necessary NodeIDs for this example were read out using "UaExpert" and permanently coded in the program code.

However, fixed NodeIDs have disadvantages and make an application very inflexible. Therefore, an application should always determine the required NodeIDs using UA_TranslatePath().

Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.

UA_Connect_0()                // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.

UA_GetNamespaceIndex_BC()    // NamespaceUri := 'http://br-automation.com/OpcUa/BC/io-system/'
// NamespaceIndex := 2; ... Index determined by FB

3) The necessary NodeIDs are only permanently coded for this example.

UA_OBJ_BC_MethodSet_ID  := '4';  // '4' MethodSet
UA_OBJ_BC_Reboot_ID     := '240'; // '240' Reboot

4) Request a method handle. To do this, you need the NodeIDs of the directory from which the method is to be called later and the NodeID of the method itself. In this example, this is method "Reboot".

UA_MethodGetHandle_0()      // Allocate Handle for Method "Reboot"

5) Calling method "Reboot"

UA_MethodCall_0()           // CALL Method <Reboot>

6) Close the connection, see example "UA_Disconnect()" on page 73.

UA_Disconnect_0()
7) Wait 1 to 2 seconds until a restart is performed.
8) Establish a new connection, see example "UA_Connect()" on page 71.

    UA_Connect_0() // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'

9) Wait until the connection has been re-established, see example "UA_ConnectionGetStatus()" on page 73.

    UA_ConnectionGetStatus()

10) Executing the cyclic application program

    Cyclic_Programm() // do application

11) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.

    UA_NodeReleaseHandle_0()

12) Close the connection, see example "UA_Disconnect()" on page 73.

    UA_Disconnect_0()

Sample program

(* Hardcoded NodeIDs *)
UA_OBJ_BC_MethodSet_ID := '4';                          // '4' MethodSet
UA_OBJ_BC_Reboot_ID     := '240';                        // '240' Reboot

(* Node <MethodSet> as Reference for method <Reboot> *)
iObjectNodeID_MethodSet.NamespaceIndex := NamespaceIndex;
iObjectNodeID_MethodSet.IdentifierType := UAIdentifierType_Numeric;
iObjectNodeID_MethodSet.Identifier     := UA_OBJ_BC_MethodSet_ID;       // MethodSet

iObjectNodeID_Reboot.NamespaceIndex    := NamespaceIndex;
iObjectNodeID_Reboot.IdentifierType    := UAIdentifierType_Numeric;
iObjectNodeID_Reboot.Identifier        := UA_OBJ_BC_Reboot_ID;         // Method Reboot

(*-- Allocate Method Handle --*)
UA_MethodGetHandle_0.ObjectNodeID  := iObjectNodeID_MethodSet;
UA_MethodGetHandle_0.MethodNodeID  := iObjectNodeID_Reboot;

UA_MethodGetHandle_0(Execute       := ExecuteMethGetHandle_0,
                ConnectionHdl := ConnectionHdl,
                Timeout       := T#10s);

//iInputArguments[0..9] ... have to be set 0
//iOutputArguments[0..9] ... have to be set 0

(*-- Call Reboot --*)
UA_MethodCall_0(Execute         := ExecuteMethCall_0,
                ConnectionHdl   := ConnectionHdl,
                MethodHdl       := UA_MethodGetHandle_0.MethodHdl,
                InputArguments  := iInputArguments,
                OutputArguments := iInputArguments,
                Timeout         := T#10s);

(*-- Release Method Handle --*)
UA_MethodReleaseHandle_0(Execute       := ExecuteMethReleaseHandle_0,
                ConnectionHdl := ConnectionHdl,
                MethodHdl     := UA_MethodGetHandle_0.MethodHdl,
                Timeout       := T#10s);
10.4.2 Switching to packed format

Channels of digital input or output modules can be accessed individually in Boolean format or – in packet format – to all I/O data points together. In this use case, the access mode for the outputs of module X20DO9322 should be switched to packed format on slot 34.

**Notes**

- For access in packed format, a variable adapted to the number of channels must be created. For details, see "Notes about digital I/Os in PACKED format" on page 29.
- The configuration for packed format Packed is set via node "DigitalOutputsPacked" or "DigitalInputsPacked". By default, the format of digital I/Os is set to "Unpacked".
- Access should take place via directory ParameterSet since all available objects are listed and can therefore be accessed via the same browse path.
- To determine NodeID "DigitalOutputsPacked" using UA_TranslatePath(), the appropriate BrowseName must be specified. For details, see "Difference BrowseName and DisplayName" on page 19.
- For information about permanently saving the configuration, see "Apply" on page 56.
  - Evaluate the returned ConfigResult from method "Apply" or the configuration status as to whether a restart is required. For more information, see "Calling methods" on page 53.
  - Method "ApplyChanges" also permanently defines the currently connected module type on the slot in the configuration. For more information, see "Fixed-position modules" on page 50.
- For information about restarting the bus controller, see "Detecting a necessary restart" on page 46.
- For information about the configuration status, see "Evaluating the configuration status" on page 47.

**Changing the format using UaExpert**

In UaExpert, value "Value" is set from 0 (off) to 1 (on) and then permanently applied to the bus controller using method "Apply".
Procedure

1) Establish a connection, see example "UA_Connect()" on page 71.

   UA_Connect_0()                // ServerEndpointUrl := 'opc.tcp://192.168.1.1:4840'

2) Get the NamespaceIndex, see example "UA_GetNamespaceIndex()" on page 75.

                                  // NamespaceIndex := 2; Index determined by FB

3) Determine the StartNodeID.

   StartNodeID.Identifier is "Root": "0:Root" = "84"

4) Determine the relative path names.

   ° Definition of the relative path name

      Path := '/0:Objects /2:DeviceSet /2:X20BC008U /2:ParameterSet'
      UA_TranslatePath();

   ° After correct execution of UA_TranslatePath(), apply the determined NodeID as the StartNodeID.

   ° Definition of relative path names

      Path_Param := '/2:DigitalOutputsPacked',
      Path_Status := '/2:ConfigurationStatusAll',

   ° After correct execution of UA_TranslatePath(), temporarily store the determined NodeID or specify
     the determined NodeID to subsequent functions.

5) Reserve the NodeID.

   UA_NodeGetHandle_0()         // UA_NodeGetHandle_0.NodeHdl;

6) Read out the current value.

   UA_Read_0()                  // Variable_Rd_0 := '::OpcTest:IO_FormatPacked';

7) If necessary, change the required data format, write in packed format.

   UA_Write_0()                 // Variable_Rd_0 := '::OpcTest:IO_FormatPacked';

8) Evaluate the configuration status to check whether another method call such as Apply or Reboot is required.

9) Save and apply the data format change using method Apply.

   UA_MethodGetHandle_0()       // Allocate Handle for Method Apply
   UA_MethodCall_0()            // CALL Method <Apply>
   UA_MethodReleaseHandle_0()   // Release Method Handle

10) Evaluate the configuration status, see "Bus controller configuration" on page 43.

     As soon as the change of data format has been applied using Apply, the corresponding X2X module is
     automatically restarted if necessary. During this time, no data exchange with the I/O module is possible,
     and read or write accesses cause an error message.

     Application tip

     Since restarting the I/O module can take different amounts of time depending on the module type or com-
     plexity of the required changes, the best solution is to check the configuration status.

11) Executing the cyclic application program

    Cyclic_Programm()           // do application

12) Free up the handle, see example "UA_NodeReleaseHandle()" on page 81.

    UA_NodeReleaseHandle_0()

13) Close the connection, see example "UA_Disconnect()" on page 73.

    UA_Disconnect_0()