NETx AUTOMATION

NETx BMS Server

System documentation

Member of: KNX Association | OPC Foundation
BACnet Interest Group Europe

Document Version: 2.0.15

Contents

Copyright

1. Product Description

NETx BMS Server 2.0

The aim of the NETx BMS Server is to solve the problem that arises when heterogeneous building automation systems are used. To achieve this, the NETx BMS Server collects data and information from the field level of the building automation system using different fieldbus technologies. In NETx BMS Server 2.0 this data can originate from KNX, Modbus, or Building Automation and Control Networking Protocol (BACnet) networks. In addition, connections to other systems like Fidelio or to foreign systems that already provide an OPC connection are possible.

Once the data is available within the NETx BMS Server, management clients can access the data through the provided management interfaces. The NETx BMS Server provides interfaces to NETx Voyager clients, to Web-based NETx BMS Client clients as well as to any other third-party client. Thus, it builds a bridge between the field/automation level and the management level of building automation systems.

1.1. Main Criteria

- More than 100.000 datapoints are supported depending on the system's hardware
- Support for various fieldbus technologies (KNX, BACnet, Modbus, JSON)
- Multiple management client interfaces (OPC Clients, BMS Clients, or Web Clients)
- Interfaces to the NETx BMS Client visualizations and to Metering and Reporting System (MaRS)
- Server-based calendar for defining timer-based calendar events directly within the server
- Metering module for analyzing smart metering devices included
- Cluster module included that provides the opportunity to integrate foreign OPC DA and/or NETx BMS Servers.
- Interconnecting multiple NETx BMS Servers in a hierarchy using clustering.
- MS SQL database included that stores historical values of datapoints
- Nearly all officially used data types of KNX, BACnet, and Modbus are supported
- High reliability and availability
- Scales to large system – usable for smart homes up to large commercial buildings
- High data transfer rate
- Redundancy by hot standby main- and backup server configuration
- Device availability checking
- Multi-project kernel
- Workspace management – multiple workspaces can be administrated
- V-Links for implementing gateway functionality
- Virtual Items for specifying virtual datapoints
- Task definition table for linking of server items and executing LUA scripts in live mode
- LUA Script Language Engine for programming own control functionality with script editor
- Event Processor – cyclic, time and event based actions can be defined
- NETx BMS Server Studio 2.0
- Custom server items can be defined to add virtual datapoints
- Data recording in the queue buffer – up to one million sent or received telegrams
- Current state of the whole system can be saved and will be loaded automatically when starting
- Online check of the consistency of all connected gateways and devices
- Real-time view of the telegram traffic with clear text description and further information
- Geo Data Interface for sun position, sunrise, sunset, age of the moon
- Several extension modules available like SQL Interface for Microsoft SQL Databases, Micros Fidelio Hotel Information Interface ...

1.2. Disambiguation

1.2.1. Project/Workspace

A NETx BMS Server project contains all the information that is necessary to collect the data and information from the building automation systems. This includes on the one hand the configuration data that is required to get the data from the fieldbus systems and on the other hand runtime information like logging data. The sum of configuration and runtime files as well as the directories that include these files of a dedicated project are referred to as workspace. The workspaces are located within the following directory:

```
<Install Directory>\NETxAutomation\NETx.BMS.Server.2.0\Workspaces
```

For example, within Windows 8.1 64 Bit, the default path is:

```
C:\Program Files (x86)\NETxAutomation\NETx.BMS.Server.2.0\Workspaces
```

! It is recommended that the workspaces are backuped in regular intervals. To do so, the whole contents of the workspaces directory mentioned above has to be copied to the backup medium.

1.2.2. Datapoint

A datapoint is a single data element within the field level of the building automation system. This can be a sensor value (e.g. status of a light switch, temperature value), an actuator value (e.g. set value of a blind, control value of a heating system) but also so called virtual datapoints that are only present within the server (e.g. a set value for the room temperature, the current date, the current sun position).

1.2.3. Server Item

A server item is the representation of a datapoint within the NETx BMS Server. A server item consists of several properties that represent the datapoint. Important examples are the value of the datapoint, the quality as well as the engineering units. However, other meta-data may also be included (e.g. range of a datapoint, alarm status).

1.2.4. Server information model/Information model

The server information model is the internal view of the data and information of the building automation systems. In more detail, the server information model is organized as a hierarchy that consists of all server items as well as further information (e.g. type information).

1.2.5. Server Item Tree

The representation of the server information model is referred to as server item tree.

1.2.6. BMS Client

A BMS Client refers to a client application that connects to the NETx BMS Server to access the Server Items. A BMS Client can be NETx Voyager client, a NETx BMS Client, or any other third-party OPC client.

1.2.7. VNET

VNET is a proprietary network protocol that can be used by management clients as an alternative to OPC. VNET is available for all NETx management clients.

1.2.8. NETx BMS Client

A NETx BMS Client is a NETx client that uses either a HyperText Transfer Protocol (HTTP) connection (for Web-based clients) or a VNET connection (e.g. for WinCE clients) to get the data from the NETx BMS Server.

1.3. System Overview

Modern building automation systems are distributed systems where the control functionality is spread across a network. Due to the differing requirements of these systems, there is no single technology that can be used to satisfy all needs. As a result, building automation systems are extremely heterogeneous where many different network technologies and communication standards are used.

The aim of the NETx BMS Server is to solve this problem that arises when heterogeneous building automation systems are used. To achieve this, the NETx BMS Server collects data and information from the field level of the building automation system using different fieldbus technologies. In NETx BMS Server 2.0 this data can originate from KNX, Modbus, or BACnet networks. In addition, connections to other systems like Fidelio or to foreign systems that already provide an OPC connection are possible. For the latter one, the Cluster module has to be used that maps the OPC items from the third-party OPC server into the information model of the NETx BMS Server. Furthermore, due to the flexible and modular design of the NETx BMS Server, an integration of interfaces to other fieldbus technologies that are not available in the current version of the NETx BMS Server is possible. Figure 1.1 shows the basic architecture of the NETx BMS Server.

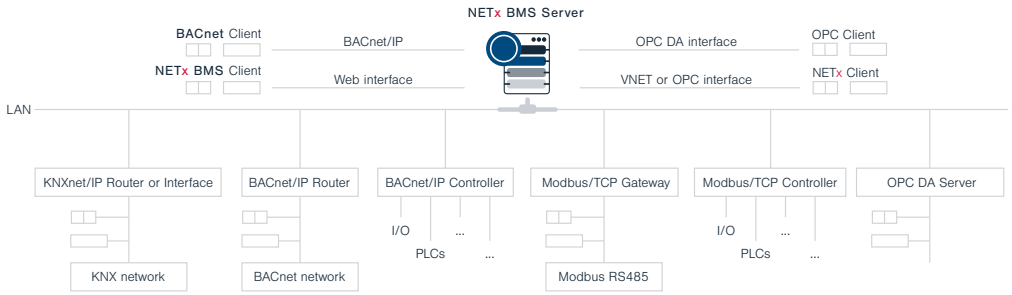


Figure 1.1.: System structure NETx BMS Server 2.0

1.3.1. System Architecture

The system consists of three main parts: the server, a Web server, and the NETx BMS Server Studio. The server is an autonomous application that implements core functionality. The Web server is service that is connected to the NETx BMS Server and provides the interface to the Web based clients. Finally, the NETx BMS Server Studio is the management interface that is used configure and maintain the server component and its workspaces. It includes useful tools which assists the administrator to managing the overall system including the NETx BMS Client clients and their visualization projects.

In Figure 1.2, the basic architecture of the overall NETx BMS Server system is shown.

A huge variety of integration is possible within NETx BMS Server 2.0.

From the field level, datapoints and devices from KNX, Modbus, and BACnet as well as from other systems like Micros Fidelio Hotel Information, Protel Hotel Information or VingCard door access system can be integrated within the NETx BMS Server. Furthermore, it is possible to integrate data from third-party OPC server using the Cluster module. The server retrieves all the information from the field level and stores it within its server information model. Within this model, the data is represented in a transparent, technological-independent way. This means that once the data is available as server item within NETx BMS Server, the underlying fieldbus technologies does not matter anymore – for a management client that accessed the information through the NETx BMS Server, the data is simply a server item and so it can handle it in a well-defined way. This has the aim that developers of management clients do not need to deal with characteristics of the underlying fieldbus technology.

Once the data is available through these generic server items, management clients can assess them through the provided management interfaces. The NETx BMS Server provided interfaces to the following management clients:

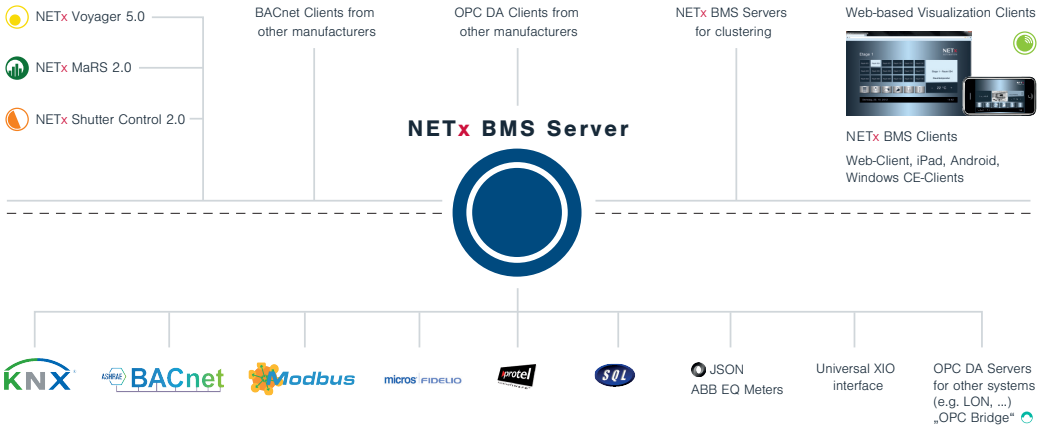


Figure 1.2.: System overview NETx BMS Server 2.0

- NETx Voyager clients that use OPC DA or VNET connection
- NETx BMS Clients that use a HTTP connection to the Web server or a VNET connection to communicate with the server
- Third-party OPC clients using OPC DA 2.05 connection
- Third-party BACnet clients using the BACnet Server interface

Depending on the operating system where a client is running, the following NETx BMS Clients are available:

- NETx Touch iOS Version: free client App for Apple's iPhone, iPad or iPod touch that can be installed via Apple's App Store.
- NETx Touch Android Version: free client App for Android devices (smart phones, tablets) that can be installed via Google Playstore.
- NETx Web Voyager Client is a client that uses the Web server of the NETx BMS Server for accessing the visualization. In fact any arbitrary Web browser supporting HTML 5 and JavaScript can be used – the used web engine within the NETx BMS Server does not required any browser plugins and so no additional software needs to be installed at the client side. Supported Web browsers are the commonly used ones.

Visualization projects for these clients are created and maintained within NETx BMS Server Studio using the so called NETx BMS Client Editor. This editor is the same for all three different kinds of NETx BMS Clients – therefore one visualization project can be used for all different client platforms at once. These clients are advantageous for smaller visualization projects where a large amount of NETx BMS Client devices have to be controlled and maintained centrally.

For large central visualization tasks, the NETx Voyager is available which can be connected via OPC or VNET. NETx Voyager is a PC-based, high-end visualization which provides more functionality that NETx BMS Clients. Typical examples are Alarm- and Event handling, task definition within the visualization, and scene control. NETx Voyager is a stand-alone application that has to be purchased separately.

1.3.2. The Server Item Tree

The basic principle of the NETx BMS Server is to get the data and information (i.e. datapoints) from the fieldbus and to store within the server as so called *server items*. Thus, a server item represents a single datapoint within the field level of the building automation system. Each server item consists of several properties that represent the datapoint within the building automation systems.

These server items are hierarchically arranged in a tree-like view called server item tree. This server item tree consists of the server items (leaves of the tree) as well as so called branches (nodes between the leaves and root of the tree) that are used to organize the tree. The server item tree can be seen as the network-visible representation of the server's information model that can be accessed by the management clients. Figure 1.3 shows an example of such a server item tree within the NETx BMS Server Studio.

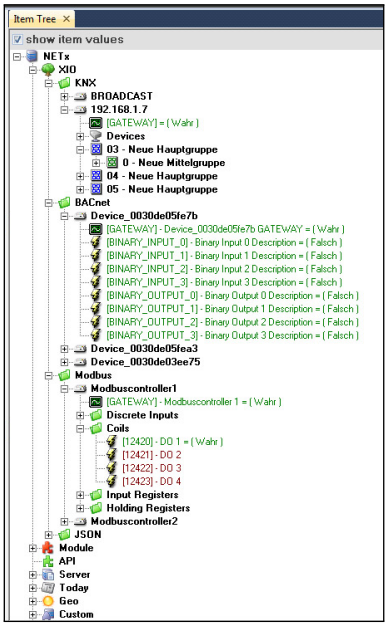


Figure 1.3.: Server Item Tree

The basic structure of this server item tree is unique within the NETx BMS Server. Server items and branches are addressed by clients using their item ID that is unique within the server item tree. This item ID refers to the browsing path that the item or branch has within the server item tree i.e. it is the concatenation of all item and branch names from the item or branch to the root node of the tree including the deliminators between the names[1]. The server item tree starts with the root node that has always the name:

`NETx\`

Within the first level of the server item tree, following branches are available:

`NETx\XIO\`

This sub-tree contains datapoints from the field level that are controlled by server. It has the following branches:

- `NETx\XIO\KNX\`: Here, the different KNX devices, router, and group addresses can be found (cf. Section 4.3.1). The following items are located within this sub-tree:
  - KNXnet/IP routers and interfaces: `NETx\XIO\KNX\<Router>\`
  - KNXnet/IP router and interfac monitoring item: `NETx\XIO\KNX\<Router>\GATEWAY\`
  - KNX group addresses: `NETx\XIO\KNX\<Router>\<GroupAddress>\`
  - KNX devices: `NETx\XIO\KNX\<Router>\Devices\IndividualAddress\`
- `NETx\XIO\Modbus\`: Here, the different Modbus devices and datapoints can be found (cf. Section 4.3.2). The following items are located within this sub-tree:
  - Modbus devices: `NETx\XIO\Modbus\<DeviceName>\`
  - Modbus devices monitoring item: `NETx\XIO\Modbus\<DeviceName>\GATEWAY\`
  - Modbus discrete inputs: `NETx\XIO\Modbus\<DeviceName>\Discrete Inputs\<Address>\`
  - Modbus coils: `NETx\XIO\Modbus\<DeviceName>\Coils\<Address>\`
  - Modbus input registers: `NETx\XIO\Modbus\<DeviceName>\Input Registers\<Address>\`
  - Modbus holding registers: `NETx\XIO\Modbus\<DeviceName>\Holding Registers\<Address>\`
- `NETx\XIO\BACnet\`: Here, the different BACnet devices and objects can be found (cf. Section 4.3.3). The following items are located within this sub-tree:
  - BACnet/IP devices: `NETx\XIO\BACnet\<DeviceName>\`
  - BACnet/IP routers: `NETx\XIO\BACnet\<RouterName>\`
  - BACnet non-IP devices: `NETx\XIO\BACnet\<RouterName>\<DeviceName>\`
  - BACnet devices monitoring item: `NETx\XIO\BACnet\...\GATEWAY\`
  - BACnet objects: `NETx\XIO\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\`
    For read-only objects:

[1] The deliminator can be configured within the server. For the rest of this documentation the default deliminator "\" is used.

...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>\
For writable objects:
Reading:...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>\
Writing:...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>-PRIO <nr1>\
Resetting:...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>-PRIO <nr1> RESET\
Writing:...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>-PRIO <nr2>\
Resetting:...\BACnet\{<RouterName>}\<DeviceName>\<ObjectName>\<DatapointValue>-PRIO <nr2> RESET\
...

- NETx\XIO\JSON\: Here, the different JSON devices and datapoints can be found (cf. Section 4.3.4). The following items are located within this sub-tree:
  - JSON gateways: NETx\XIO\JSON\<GatewayName>\
  - JSON gateway monitoring item: NETx\XIO\JSON\<GatewayName>\GATEWAY\
  - ABB EQ Meter: NETx\XIO\JSON\<GatewayName>\<Serial>\
  - ABB EQ Meter monitoring item: NETx\XIO\JSON\<GatewayName>\<Serial>\GATEWAY\
  - ABB EQ Meter energy values:
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\importenergy\
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\importenergy\total\
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\importenergy\l[1-3]\
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\importenergy\t[1-4]\
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\exportenergy\
    ...
    NETx\XIO\JSON\<GatewayName>\<Serial>\apparent\netenergy\
    ...
    NETx\XIO\JSON\<GatewayName>\<Serial>\reactive\
    ...
    NETx\XIO\JSON\<GatewayName>\<Serial>\active\
    ...

NETx\Cluster\
Within this sub-tree, items of the Cluster module (e.g. items from foreign OPC DA Servers, items from other NETx BMS Servers) are located.

- NETx\Cluster\VNET\
  Here, the different Server Items of integrated, remote NETx BMS Servers can be found (cf. Section 4.4.1). For each NETx BMS Server, the sub-tree NETx\Cluster\VNET\<IPAddress>\<ServerName>\ is created. The following items are located within this sub-tree:
  - NETx\Cluster\VNET\<IPAddress>\<ServerName>\Connected\: This item indicates whether the connection to the NETx BMS Server is working.
  - NETx\Cluster\VNET\<IPAddress>\<ServerName>\<ItemName>\: In addition, all Server Item that shall be integrated from the remote NETx BMS Server are listed.
- NETx\Cluster\OPC_DA\
  Here, the different OPC Items of foreign OPC DA Servers can be found (cf. Section 4.4.1). For each OPC DA Server, the sub-tree NETx\cluster\OPC_DA\<IPAddress>\<ServerName>\ is created. The following items are located within this sub-tree:
  - NETx\Cluster\OPC_DA\<IPAddress>\<ServerName>\Connected\: This item indicates whether the connection to the OPC DA Server is working.
  - NETx\Cluster\OPC_DA\<IPAddress>\<ServerName>\<ItemName>\: In addition, all OPC items that shall be integrated from the foreign OPC DA Server are listed.

NETx\Module\
Within this sub-tree, items of NETx BMS Server modules are located. Currently, the following module is available:

- NETx\Module\MaRS\
  Here, the different resources and meters of the Metering module can be defined (cf. Section 4.4.2).

NETx\API\
Within this sub-tree, items of additional driver modules (e.g. items from the Fidelio module) are located.

`NETx\Server\`
Here, information about server is stored (e.g. status of server, number of connected clients, . . .). Within the sub-tree `NETx\Server\Database\` the status of database connection can be seen.

`NETx\Today\`
Within this sub-tree, time and date as well as weather information can be retrieved.

`NETx\Geo\`
Here, geographic information like sunset time and moon age is available.

`NETx\Custom\`
Using LUA scripts, it is possible to create so called custom items that can be seen as virtual items that are only available within the server (cf. Section 4.6). It is also possible to define them within a user-specific hierarchy. These custom items are located within this sub-tree.

`NETx\Aliases\`
It is possible to define aliases i.e. alternative names for server items. These aliases can be found here.

`NETx\VAR\`
This sub-tree contains pre-definied, virtual datapoints that can freely be used by clients.

`NETx\XCOMMAND\`
This sub-tree contains the definition of the so called XCOMMANDs that are used by the server-based event modules (for future use).

`NETx\VIRTUAL\`
Here, the virtual items that are defined within the current workspace are located. According to the configuration, these items can be arranged within a hierarchy.

`NETx\XCON\`
Here, different items that can be used for communication with other systems or network services are located (e.g. items for UDP, TCP, HTTP, COM port, e mail communication).

2. Getting Started

2.1. Installation

Insert the disk into the optical drive.

Either the installation software will start at autorun or the user will need to execute it manually.

The setup will guide the user through the installation. During this process it is possible to make a few specifications such as a different installation path than the default one.

Inside the directory – either specified by the user or the default one – a sub folder “Workspaces” will be created. The latter one will not be deleted when deinstalling the software. It has a special role in saving workspaces and also contains the default workspace, from which all new workspaces will derive.

! Note that under Microsoft Windows™ server systems the installation of Microsoft .NET Framework 3.5™ might need to be done manually by installing the according server feature.

!Attention: All anti virus software and anti mall ware has to be deactivated during installation. The user installing the software has to have appropriate rights to do so. Make sure the necessary exceptions are set in the security software so that NETx BMS Server is working properly. For compatibility reasons it is also not recommended to run NETx BMS Server parallel to NETx Voyager Server 1.0 on the same machine.

With the NETx BMS Server several different components are installed and registered on the system – the NETx BMS Server itself as a service or server component, the web server as a service to provide NETx BMS Clients with the necessary information, the NETx BMS Client Editor as an executable program, the NETx BMS Server Studio 2.0 as an executable program and different drivers to support BACnet, Modbus, KNX, and VNet as communication interfaces. Microsoft SQL Server Express™ is also installed to store historical data.

2.2. First Steps

First start the NETx BMS Server Studio 2.0. This can be accomplished by either clicking the icon on the desktop or one can run the program out of the start menu. When the software is run the first time and there is no valid hard-lock (USB stick) connected to the system, a warning will appear in the system messages that this is an unlicensed copy of the software and the demo mode will continue to work. To license the software either a valid hardware lock can be inserted or the License Manager has to be run. For first steps the evaluation mode is sufficient. For more detailed information about licensing the software, please refer to section A.2.

Again assuming the software is run the first time it will start the “MyFirstWorkspace” or the “Default” workspace. Once a new one is created and the program is closed, it will reopen with the last active one at startup.

! It is not recommended to work in the “Default” workspace since all future workspaces created will use it as template. All properties set in it will be inherited by the new workspace.

To create a new workspace, go to the menu “Workspace” and choose “New Workspace”. The program will now prompt the user with a mask to put in the new workspace name. Since the workspace name will also be used as folder name for the workspace it is not allowed to use special characters like quotation marks, slashes, backslashes, or similar.

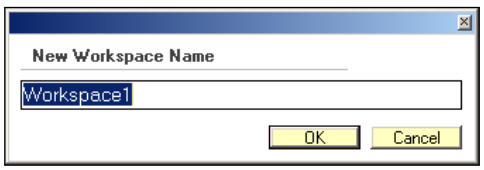


Figure 2.1.: Name new workspace

Should the server have not been shut down already, it will shut down now and an attempt will be made to start the newly created workspace. It contains a small demo project which is present in the “Default” workspace.

The next step to get the system up and running would be to include the configuration data of the devices and datapoints that are used at the field level of the building automation system.

2.2.1. KNX Configuration

The import for KNX group addresses is a quick way to add many group addresses at once. Open the menu item “KNX” and select “Import ETS© Project. . . ”.

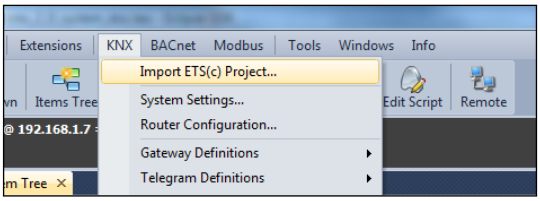


Figure 2.2.: Open ETS Converter

A dialogue will appear (cf. Figure 2.3) where on first line the file to import can be selected. The second important field is “Default Gateway” which determines the gateway the telegrams will be sent to for all the group addresses which will be imported this time. For the first steps it is suggested to type in the IP address of the KNXnet/IP router e.g. “192.168.1.70”. Check the boxes to create a telegram definition file, link definition file, and a gateway definition file and leave all other entries on their default values. Click “Convert” and after confirming the creation of the definition files close the mask.

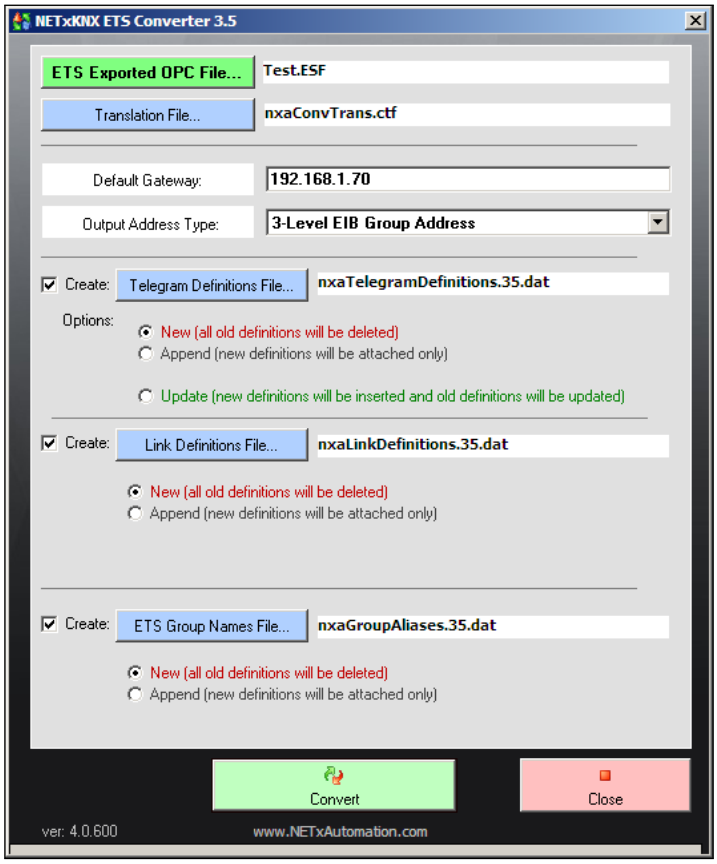


Figure 2.3.: Convert ETS File

2.2.2. Modbus Configuration

If there are Modbus devices connected to the system please follow the next steps. First open the device definition

table under the menu item “Device Definitions” within the “Modbus” menu (cf. Figure 2.4).

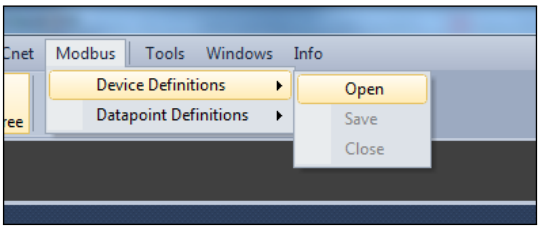


Figure 2.4.: Open Modbus Device Definition

Now either edit a line in it or add a new one using the according command of the context menu (cf. Figure 2.5). The first field “Device Name” is a unique name which can be freely chosen, the “IP Address” of the device needs to be entered in the second field and the “Port” to connect to in the third one. If the device is a native Modbus/TCP device “0” has to entered within the next field – otherwise the “Slave ID” has to be typed in. In the fifth field a description for the device can be added. All following fields are optional and therefore will be described in detail in a later chapter.

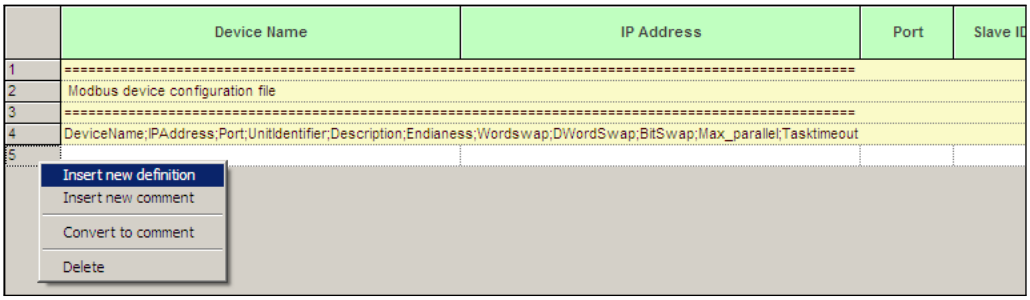


Figure 2.5.: Edit Modbus Device Definition

For editing the actual datapoints please open the according definition file (menu “Modbus”, menu item “Datapoint Definitions”). The “Device Name” needs to be identical to the one which has been defined previously. It represents the reference to the device that holds the actual datapoint. “Modbus DP Type” defines the datapoint type inside the Modbus device. The appropriate one needs to be selected. “Address” contains the memory address within the Modbus device where the datapoint is situated. “Data Type” defines which kind of data value the memory cell in the Modbus device contains. Should the data type be a “String” or “WString”, “Size” determines the length of it. All other data types define their length on their own and the field should be set to “1”. Enter a short “Description” of the datapoint. “Polling Interval” contains the value in milliseconds of the interval the data will be requested in from the device. Again all further fields are optional and therefore will be described in detail in a later chapter.

2.2.3. BACnet Configuration

The quickest way to include a correct BACnet configuration is to run the BACnet Explorer and save its output to the definition files.

If the BACnet Explorer is run the first time the user will be prompted with the message that “127.0.0.1” is not a valid address and that it should be changed. For now “Yes” would be a good choice. Now please select the correct IP address using the next dialogue. Confirm with “OK” and the actual BACnet Explorer will open and scan the network for available devices.

Check any of the device’s boxes and the datapoints will be scanned and included into the configuration. Click the “Export” button to write the data to the configuration files. After confirming the creation of the files the user can close the BACnet Explorer with the “Exit” button.

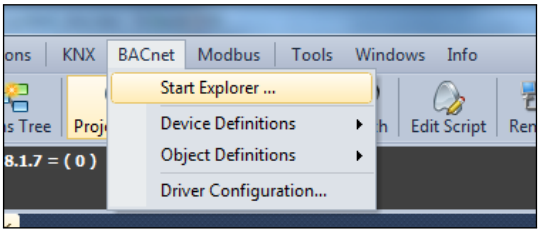


Figure 2.6.: Open BACnet Explorer

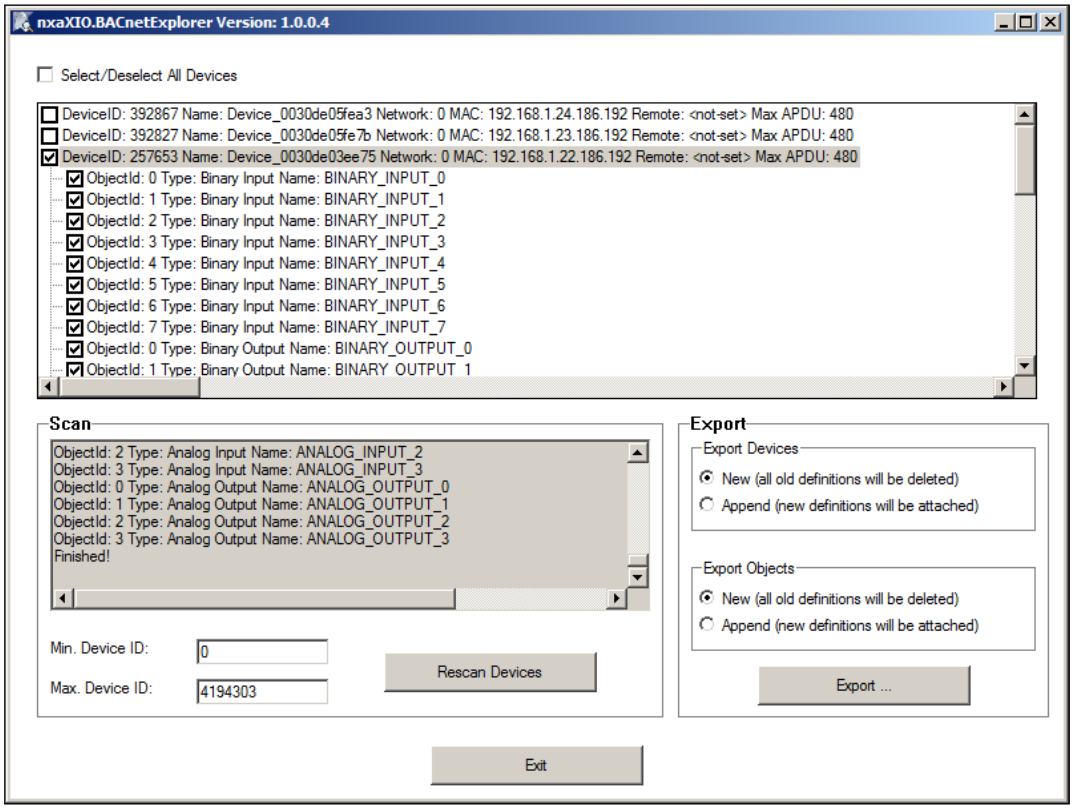


Figure 2.7.: BACnet Explorer

2.2.4. Starting the Server

To start the server click the green arrow in the toolbar. Some log information will be shown in the “System Messages” and then the “Item Tree” shows up. Under node “XIO” the devices can be found in their according category. When the devices there are opened, the datapoints that are assigned to the devices are shown. Using the menu item “Send Telegramm . . . ” or “Write Item Value . . . ” within the context menu (right-click on the datapoint), the values of the datapoints can be changed when they are writable. In addition, the current datapoint value and the corresponding item properties can be shown within “Properties” window at the right hand side.

2.2.5. Visualization Project

To create a new visualization project, right click “Visualization Project” inside the “Project Tree” and Choose “New visualization project. . . ”.

The user will be asked to put in the name of the new NETx BMS Client project. Afterward the NETx BMS Client Editor will open and the user will be able to design and edit the NETx BMS Client project.

Once the NETx BMS Client Editor is closed the project can be reopened for editing by either double-clicking the project’s name in the “Project Tree” or by the context menu’s item “Edit visualization project. . . ”. Once changes

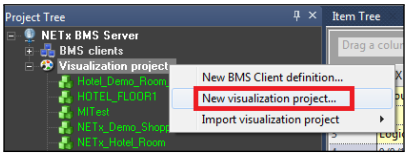


Figure 2.8.: Create a new Visualization Project

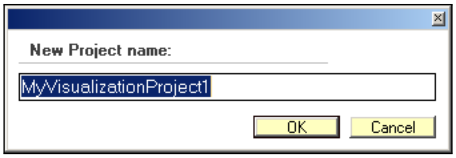


Figure 2.9.: Name the new visualization project

have been made to the project, it is necessary to push the project to the NETx BMS Clients. This can be achieved by the menu item “Push visualization project to BMS Clients. . . ” from either the context menu or the menubar / “Project”. Visualization projects shown greyed in the project tree do not have a linked workspace in the NETx BMS Client Editor and therefore have to be updated manually.

2.2.6. BMS Client

The NETx BMS Clients can be created very easily. A right-click to “BMS Clients” within the “Project Tree” will open a context menu from which “New BMS Client definition. . . ” can be chosen.

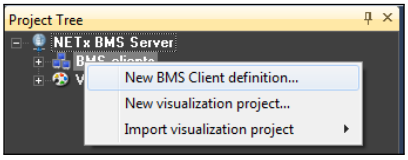


Figure 2.10.: Create new BMS Client definition

A dialog will appear and the specifications to the BMS Client can be made. The field “Name” has to be filled in to name the client. Set “Enabled” to true to be able to connect to the client. “Project” determines which of the NETx BMS Client projects shall be displayed with this BMS Client. Other fields are not mandatory and can be left empty.

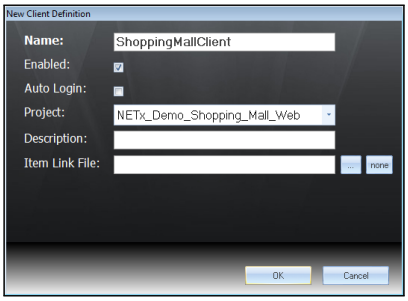


Figure 2.11.: BMS Client Properties

3. NETx BMS Server Studio

The NETx BMS Server Studio is divided into different areas. The following short overview will describe them.

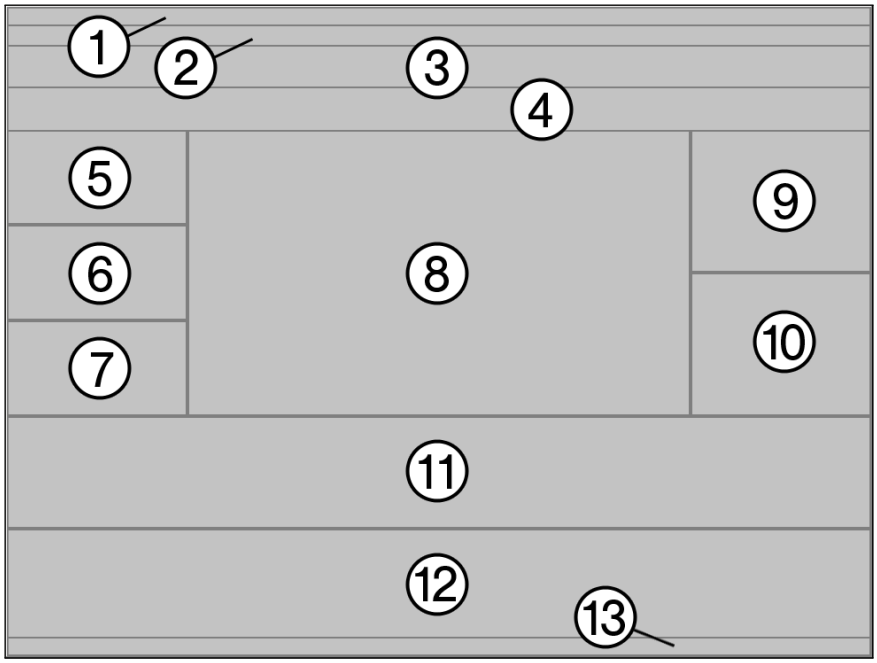


Figure 3.1.: NETx BMS Server Studio - Areas

1. Program Bar - It shows the name of the actual opened workspace and its online state.
2. Menu Bar - The user can access all menu commands here.
3. Tool Bar - Provides the most important commands in one click.
4. Info Bar - Gives the user a short overview about the server state.
5. Project Tree - Shows the visualization projects and the defined BMS Clients of the current workspace.
6. Gateways - Within this window, the actual state of the different gateways is shown.
7. Cells - Here the item values can be changed.
8. Main Working Area - The “Item Tree”, configuration tables and script files will appear here to be viewed or edited.
9. Telegrams - Incoming and outgoing telegrams to the field layer can be monitored here.
10. Properties - The properties of a server item can be viewed in this area in detail.
11. Graph - View the development of the values of any server item here.
12. System Messages - The log output of the server will be displayed in this area.
13. Status Bar - this shows compact overview about the state of the server and its components.

In the following sections the above introduced areas will be described further and in higher detail. Additional masks (e.g. BACnet Explorer or ETS Converter) can be found at the end of this section.

3.1. Menu Bar

The Menu Bar of the NETx BMS Server Studio can be split into six groups:

Figure 3.2.: NETx BMS Server Studio Menu Bar

1. Server-specific menus – Workspace, Edit, Server
2. Visualization menu – Visualization
3. Additional Connectivity – Cluster, Modules
4. Additional Functionality – Extensions
5. Bus-Specific Menus – KNX, Modbus, BACnet, JavaScript Object Notation (JSON), Simple Network Management Protocol (SNMP)
6. Studio-Specific and Administrative Menus – Tools, Windows, Info

3.1.1. Workspace

3.1.1.1. New Workspace

This creates a new workspace. The new workspace directly derives from the “Default”-workspace. All properties and preset values will be inherited. The name of it will also be used as a directory name. Therefore special characters such as colon, semicolon, slash, backslash are prohibited to use. When the workspace was created successfully the server will open it and start the workspace right away.

3.1.1.2. Open Workspace

This opens an existing workspace. The appearing dialog shows all available workspaces. The window allow the user sorting these ascending or descending and displaying them in different views. To open a specific workspace either mark it and click the [OK]-button or double click a workspace. The server will, if it has not happened yet, shut down. The most recent workspace will be closed and the newly chosen one will be opened and started right away.

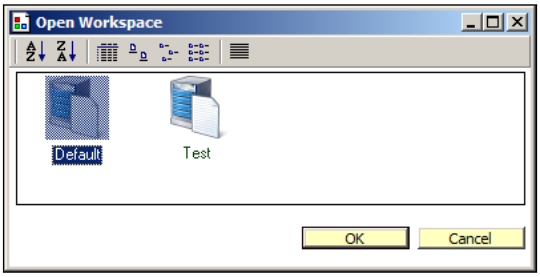


Figure 3.3.: Open Workspace

3.1.1.3. Save Workspace As

It stores the current workspace under a new name. All relevant files of the source workspace are copied into the new one. The server will be shut down and the new workspace will be loaded. Afterward the server will start the workspace immediately.

3.1.1.4. Delete Workspace

This deletes irrevocably the selected workspace with all its files, content, data, and directories. The currently opened workspace cannot be deleted.

! It is strictly not recommended to delete the “Default”-workspace since all new workspaces derive from it.

3.1.1.5. Export Workspace

This feature opens a file save dialog and exports the actual workspace to a file. The user is asked to specify a name for the export file. The default name of this file is the name of the workspace (e.g. “MyWorkspace.sew”). This “Server-Exported-Workspace” (sew) can now be sent to any other NETx BMS Server and imported there.

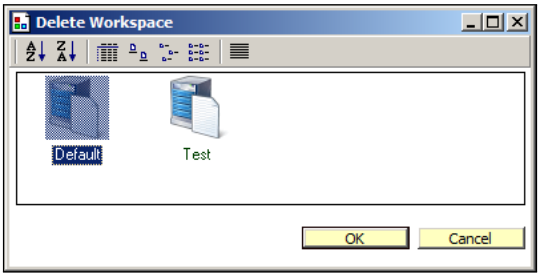


Figure 3.4.: Delete Workspace

3.1.1.6. Import Workspace

This feature opens a dialog and imports a new workspace from an export file. After selecting the file, the NETx BMS Server Studio will ask the user to specify a name for the new workspace. The server is extracting the information of the selected file and creates a new workspace with the specified name. It will be opened after the import and the NETx BMS Server will start it up.

3.1.1.7. Exit

It closes the NETx BMS Server Studio. The NETx BMS Server will remain in its actual state - either “Started”, “In Simulation”, or “Shut Down”.

3.1.2. File

All file commands are active only during administration of the content of any definition table.

3.1.2.1. Save

Saves the currently opened configuration table. The user is asked whether or not to restart the server. If yes, the NETx BMS Server will be restarted using the new table.

!Attention: Restarting the NETx BMS Server will disconnect all clients.

3.1.2.2. Save all

Saves all open definition tables. The user is asked whether or not to restart the server. If yes, the NETx BMS Server will be restarted using the new tables.

!Attention: Restarting the NETx BMS Server will disconnect all clients.

3.1.2.3. Close

Closes the currently opened configuration table.

3.1.2.4. Export to Excel...

This exports the active definition table into a Microsoft©Excel file.

3.1.2.5. Import from Excel...

The function imports the active definition table from a Microsoft©Excel file and overrides the current contents.

3.1.2.6. Print...

Prints the currently opened configuration table (e.g. “Task Definitions” or “Gateway Definitions”).

3.1.2.7. Page setup...

Opens a dialog to setup page size, orientation, and margins.

3.1.3. Edit

All editing commands are active only during administration of the content of any definition table. They are similar to editing commands of any common text editor.

- Cut [Ctrl] + [X] – It cuts marked text copies it to clipboard.
- Copy [Ctrl] + [C] – It copies marked text to clipboard.
- Paste [Ctrl] + [V] – It pastes text from clipboard to edit line.
- Search [Ctrl] + [F] – It searches for a text in the definition table.
- Search and Replace [Ctrl] + [H] – It searches for and replaces a text in the definition table.
- Select All [Ctrl] + [A] – It selects the whole text.

3.1.4. Server

3.1.4.1. System Log File

This menu entry allows you to open and close the actual system log file. The user is able to view all events in here and check upon the health of the server more detailed than in the “System Messages”.

Columns of the log file can be dragged onto the top of the table. This will group events by the content of this column. The example in Figure 3.5 shows how the column “Module” is placed on the grouping header of the table.

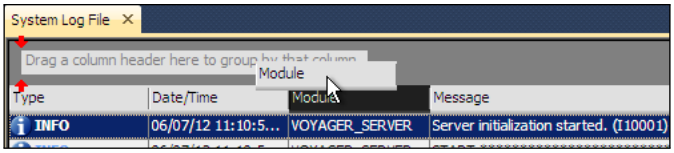


Figure 3.5.: System Log File

3.1.4.2. Server Configuration

Special server parameters can be changed in the upcoming dialog. Each option can be selected and its value can be altered. On selecting a parameter its description is shown in the box below the parameter list. For any further information about each parameter and its values it is possible to refer to section 4.2.1. Also all the parameters are listed and editable in the server configuration file.

3.1.4.3. System Configuration

Here special System Parameters can be changed. You can select each option and change its value. On selecting a parameter its description is shown to the administrator. For any further information about each parameter and its values please refer to Section 4.2.2. All these parameters are also listed and editable in the System Configuration File.

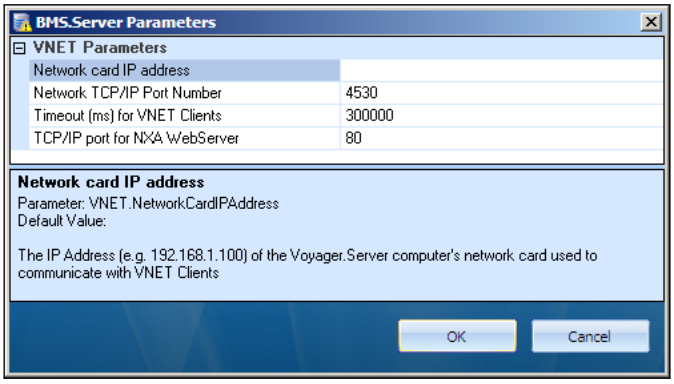


Figure 3.6.: Server Configuration

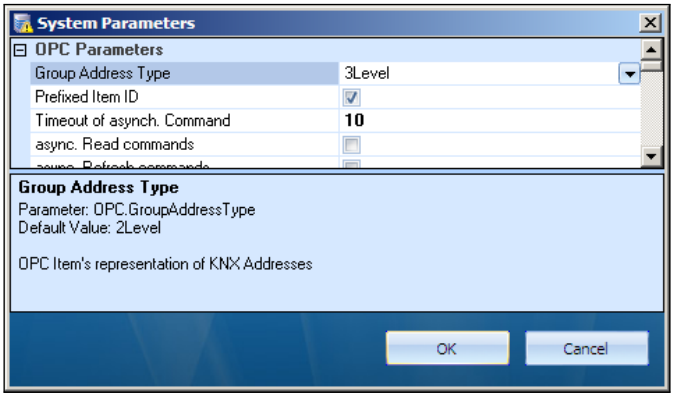


Figure 3.7.: System Configuration

3.1.4.4. N-Mesh Configuration

The N-Mesh Configuration allows you to set up a main-backup-solution for high system reliability. Simply check “Use Redundancy” and “Enable Synchronization” to activate the feature. For more detailed information about the single properties of the N-Mesh configuration refer to Section 4.2.3.

The principal of the system is as simple as it is effective. Main server and backup server synchronize their data continuously at all time. Should the main server go offline (e.g. for maintenance reason) the backup server has a complete mirror of all data. It then takes over the communication with the field bus layer. If Main Server starts again, Backup will release the fieldbus devices and the Main Server takes over control about all bus devices.

Should the connection be interrupted between Main Server and Backup for a period of time longer than the “Connection Timeout” the Backup assumes that the Main Server is down and will attempt to connect the devices of the field layer.

In combination with NETx Voyager the Main Server disconnects the visualization before it stops. Then the NETx Voyager automatically connects to the Backup Server and vice versa.

In addition to the configuration mentioned above it can be defined which datapoints will be synchronized. The definition is done in the N-MESH routing table. If the synchronization of all datapoints is necessary there is no need to configure all datapoints by hand. Check “Enable Routing” and configure the “Node IP address” of the other NETx BMS Server, where it should be routed to.

3.1.4.5. Start Server

This starts the server and all additionally installed services (e.g. Web Server). It is inactive while the server is running and will be activated, if the server is stopped or the NETx BMS Server Studio has not connected to the server yet.

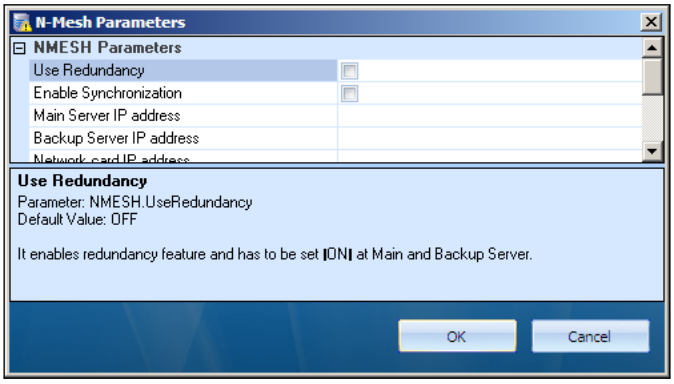


Figure 3.8.: N-Mesh Configuration

3.1.4.6. Shut Down Server

This shuts the server down. If one or more OPC Clients are still connected to the server, a dialog window will be shown to the user (cf. Figure 3.9).

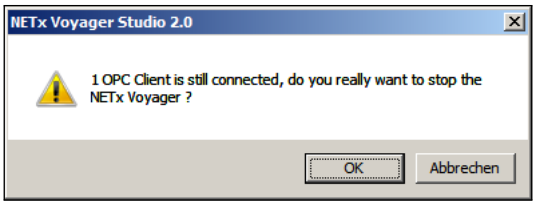


Figure 3.9.: OPC Client still connected

!Attention: Although the system has an OPC shutdown interface and the information regarding the shutdown is sent to all the clients by the help of the server, some clients do not consider this information and report a connection error!

3.1.4.7. Restart Server

With this function the system is initialized and the settings and all definition tables are loaded newly. Should the server be already running it will be shut down. After loading all settings it will be started again. In case one or more OPC Clients are still connected to the server the user will be prompted with a warning and the decision to continue with the process or not (cf. Figure 3.9).

3.1.4.8. Advanced Configuration

Extended Logging ON/OFF
Turns the extended logging on or off. If turned on a lot more log messages are written into the server log files. The log file will also grow a lot faster. Therefore this feature should be turned on in special circumstances (e.g. in a support case) only. In daily work it is recommended to stay off. When the Extended Logging is activated or deactivated an entry is made in the “System Messages”.

Set Logging Level
You can set the log level here. Like with the “Extended Logging” this feature is creating more log entries when not set to default value. In daily work it is recommended to stay on value “0”.

System Configuration File (Open/Save/Close)
Edit the system configuration file in here. The use of this feature is recommended to advanced users only.

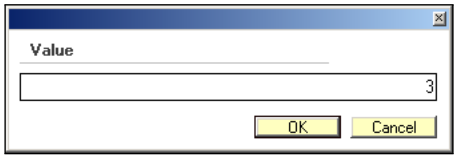


Figure 3.10.: Set Logging Level

Router Configuration File (Open/Save/Close)
Edit the router configuration file in here. The use of this feature is recommended to advanced users only.

N-Mesh Configuration File (Open/Save/Close)
Edit the n-mesh configuration file in here. The use of this feature is recommended to advanced users only.

3.1.5. Visualization

This menu is directly related to the “Project Tree”. All sub items are applied to the branches “BMS Clients” and “Visualization projects”, respectively. The Menu items are context-sensitive and are only available if the function can be use in the context of the current selection. Some items are only available if the client is online and connected to the server.

3.1.5.1. New BMS Client Definition...

Creates a new BMS Client Definition.

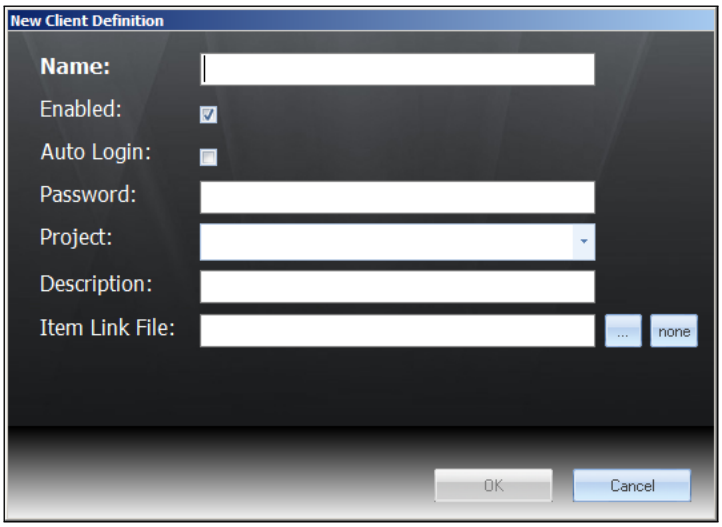


Figure 3.11.: Add Client Definition

In the “New Client Definition” Dialog you can enter the following:

- Name – Name of the NETx BMS Client
- Password – The password of the default user to access the NETx BMS Client.
- Project – The visualization project to display.
- Description – A description to this particular client definition.
- Item Link File – A translation file from one Server Item to another to be able to use the same visualization project for clients (e.g. a hotel rooms – same structure, different Server Items).

!Attention: The fields *name* and *password* have to be identical with the default user and the respective password in the visualization project.

The Auto-login feature is used in NETx BMS Client and NETx Touch to login automatically.

!Use this feature only in secured, non-public networks, because it opens the Web server for anyone!

The item Link file is used if you would like to redirect Server Items inside the visualization project to different ones. This allows the system integrator to create one visualization project which can be used several times with different Server Items. A possible example could be a hotel room. The customer would see the same visualization project in every single room. With the Item Link File underneath, the Server Items are different ones from room to room.

Once a Virtual Item Link (VIL) file is selected, it is copied from its source to the `DataFile` directory of the workspace directory.

The VIL is a text file, where in each line the first part contains the original Server Item, ";" is used as a separator, and the second part is the new item which should be used instead of the original one.

```
\NETxKNX\192.168.1.7\03/0/002;\NETxKNX\192.168.1.7\03/0/020
```

In this example the KNX group 03/0/020 will be controlled instead of 03/0/002.

3.1.5.2. Edit BMS Client Definition...

This option opens a selected and existing NETx BMS Client definition for editing.

3.1.5.3. Delete BMS Client Definition...

Deletes the selected NETx BMS Client definition.

3.1.5.4. Open BMS Client in web browser...

If a NETx BMS Client is selected in Project Tree, this menu command will start the default Web Browser and show either the login web page or will authenticate the user automatically (auto login) and proceed directly to the home page.

3.1.5.5. Disconnect BMS Client

This function interrupts the connection to an online NETx BMS Client.

3.1.5.6. New visualization project...

A new visualization project will be created and opened in the NETx BMS Client Editor.

3.1.5.7. Import visualization project / Exported Visualization Project...

A visualization project exported from NETx Voyager or NETx BMS Client Editor can be imported into the Project Tree.

3.1.5.8. Import visualization project / From other Workspace...

A visualization project from a different NETx BMS Server workspace can be imported into the Project Tree.

3.1.5.9. Edit visualization project...

The selected visualization project will be opened in the NETx BMS Client Editor.

3.1.5.10. Copy and edit visualization project...

With this function, the NETx BMS Client project will be copied and then the copied project will opened by NETx BMS Client Editor.

3.1.5.11. Delete visualization project

This deletes the selected visualization project from the Project Tree.

3.1.5.12. Push visualization project to BMS Clients...

This will push the selected visualization project to all NETx BMS Clients assigned to it.

3.1.6. Cluster

3.1.6.1. Start Explorer...

The Cluster Explorer allows the system integrator to select and integrate datapoints from alien OPC Servers or other NETx BMS Servers. The datapoints to be synchronized with this server instance can be chosen by simple mouse clicks without any previous knowledge about the internal data structure of the server to be synchronized. The Cluster Explorer can be called only while the server itself does not run similar to the BACnet Explorer.

3.1.6.2. Server Definitions

Alien servers can be connected to the NETx BMS Server either by using OPC DA or VNET. Thus data can be gathered in one central point clustering different servers of varying vendors. This has an impact on possible clients as well. Was it formerly necessary to have a client which is able to connect to several servers at once, now the spectrum of choice is a lot broader. It also is possible to create a redundant system (main / backup) with a cluster solution. Another possibility is it to install another NETx BMS Server on a distant computer, gather data there from different OPC Servers, and relay it to the central server bypassing DCOM with the VNET protocol. For more detailed information about the table's structure please refer to section 4.4.1.1.

3.1.6.3. Item Definitions

The definitions of the items have to be integrated to the letter and case-sensitively. For more detailed information about the table's structure please refer to section 4.4.1.2.

3.1.7. Modules

3.1.7.1. Metering

Meter Definition
Here you can edit the Metering definition table. After changing it it is recommended to restart the system. For any more detailed information about the single fields of the Device Definitions please refer to Section 4.3.3.2.

3.1.7.2. Fidelio/Opera Interface

Room Definitions
The table for the room definitions contains the information for an entire structure of buildings, floors, and rooms underneath the main branch FIDELIO in API. The predefined values in each room are static and will be filled by the hotel management system itself. The wake up time can be set by the customer in the room and will be written back into the Fidelio system.

Driver Configuration...
This opens the specifications for the Fidelio or Opera interface. After changing any parameters in the dialog the server needs to be restarted.

3.1.8. Extensions

3.1.8.1. XCommand event definitions

Xcommand event definitions inside the NETx BMS Server enable to link xcommand definitions to specific events on the server. For more detailed information on how to define such event definitions please refer to Section 4.5.1.

3.1.8.2. [Live] Task Definitions

Task Definitions are used to create a link between Server Items and the possibility to execute LUA scripts or to create new telegrams to the field layer. This behavior can be delayed by a fix timespan. Upon saving the NETx BMS Server loads the Task Definitions anew and executes them while remaining running. No restart is necessary. For more detailed information please refer to Section 4.5.2.

3.1.8.3. [Live] Holiday Definitions

The Holiday Definitions inside the NETx BMS Server are important for the use of the server calendar. Holidays can be excluded or explicitly included into recurring calendar events. The holiday definitions allow the administrator to define a list of holidays upon which calendar functionality cane be based. At saving the holiday definitions the NETx BMS Server loads them anew and applies them immediately while remaining running. No restart is necessary. For more detailed information please refer to Section 4.5.3.

3.1.8.4. Item Alias Definitions

It might be inconvenient to just use the hierarchy and Item IDs the NETx BMS Server provides. Declaring aliases allows the system integrator to name and sort the Server Items in any way. Different items of varying buses might be put together in a hierarchy which mirrors the structure of a building instead of the bus infra structure. So instead of KNX item 06/7/230 or BACnet item BINARY_OUTPUT_0 they may become;

```
Building01\Floor03\Room02\LIGHT_SWITCH_03
Building01\Floor03\Room02\ROOM_TEMP
```

There are no prefixes set. This means the system integrator may define his own Item IDs without any path limitations (except for the delimiter). Es sind keine Präfix gesetzt. Der Systemintegrator kann die Item ID selbst definieren ohne eine Einschränkung des Weges (Beistrich ausgenommen).

3.1.8.5. Virtual Item Definitions

The content of this table defines the virtual items under the node “VIRTUAL” in the item tree. Virtual items are not connected to any immediate bus items. Nevertheless it is possible to set these items like any other, store data inside, or even have them being persistent or historical. It is also possible for the administrator to set up LUA functions for the “set”-, “send”-, and “receive”-events of each individual virtual item.

3.1.8.6. V-Link Definitions

Virtual links attach any kind of items out of the item tree to each other. For an example; it is possible to attach a Boolean item (light ON/OFF) to an virtual integer item (zero/one) or even the other way around. Any number other than zero will be interpreted as true while zero itself will be converted to false. Strings with a length greater than zero are converted to Boolean true while an empty string is converted to false. It is also possible to convert strings into numbers and vice versa. If a non numeric string is attempted to be converted to an integer or float value an error will arise. The link is triggered when the source item is written and will transfer the value to the target item. Thus the value will be written to this one as well.

3.1.9. KNX

3.1.9.1. Import NETx ETS(c) app file...

Import an entire ETS project exported by the *NETx BMS App* for ETS from `http://knx.org/knx-en/software/ets-apps/features/?d=NETxAutomation`. The app allows you to export the ETS project including KNX Group Addresses, the deployed KNXnet/IP routers and interfaces, and the full ETS project structure including the network topology (areas, lines, devices, and Group Objects) as well as the building and trade view.

3.1.9.2. Import ETS(c) project...

This starts the importing App, which is able to convert the exported EIB Session File (ESF) in telegram- and link definition files (cf. Section 3.4.1). After each change in the definition files the NETx BMS Server has to be restarted to apply changes.

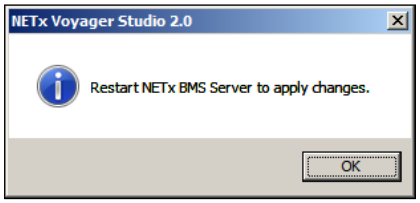


Figure 3.12.: ETS Information

3.1.9.3. System Settings...

How many send telegrams are allowed per IP Gateway in a certain amount of time? The Group Address Type is set here as well.

Send Interval
It defines the interval of telegrams being sent to a single gateway. The KNX bus can handle approximately 17 telegrams per second. The data management module of the NETx BMS Server is responsible for sending telegrams to the gateway within the defined time. If for example 2000 telegrams are sent to a gateway, it will take 2000 x 250ms = 8.3min, until all of them are sent out. We suggest calculating with 10 telegrams per second and KNX line. Then there is enough capacity left in case of occasional high loads and still no loss of any telegram to the line. The default value of 250ms is set for being able to send out 4 telegrams per second and still enough buffer to receive telegrams from the KNX line in between.

Group Address Type
This property defines how the logical KNX address is shown in the studio (in 2 or 3 steps). This parameter does not have any influence on the Server Item ID (which mainly consists of the logical KNX address). The type of the address being used for the Server Item ID is set in the file “nxaOPCSystem.35.cfg” (cf.).

3.1.9.4. Router Configuration...

You can do your KNX router definition in here. In the blue field below in the window you can find more detailed information about any parameter you can adjust. For more detailed information please refer to Section 4.3.1.1.

3.1.9.5. ETS object definitions

Here the ETS objects as shown in the item tree are defined. In general, there is no need to modify these contents since they are generated by the NETx BMS App for ETS. After changing contents it is recommended to restart

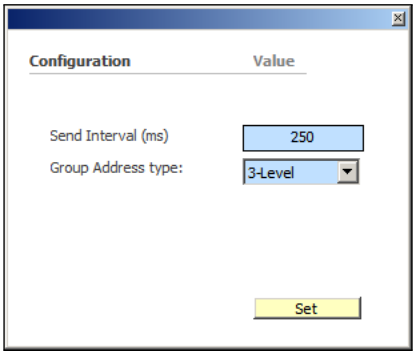


Figure 3.13.: KNX Configuration

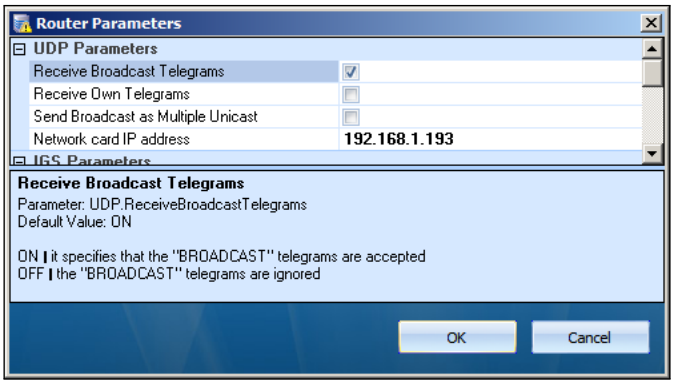


Figure 3.14.: Router Configuration

the system. For any more detailed information about the single fields of the ETS object definition please refer to Section 4.3.1.7.

3.1.9.6. Gateway Definitions

Here all used KNX NETIP Gateways have to be defined. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the Gateway Definition please refer to Section 4.3.1.3.

3.1.9.7. Telegram Definitions

This table can be created automatically with the import of an ESF file. It defines to which KNX telegrams the server should listen, how to hold their values, and if they are to be held as persistent and/or historical. It is possible to edit this table manually here. For larger changes simply it is advised to export the telegram definitions to Excel and import them back after the changes are made. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the Telegram Definition please refer to Section 4.3.1.2.

3.1.9.8. Device Definitions

This table contains the devices the system is supposed to check cyclically for their availability. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the Device Definition please refer to Section 4.3.1.4.

3.1.9.9. Group Alias Definitions

Descriptions of KNX main groups and possible sub groups can be defined in this table. The names will be used as value for the branches and sub branches in the KNX part of the Item Tree and can be read via OPC and VNET.

3.1.9.10. [Live] Link Definitions

This links logical group addresses together as it is usual in the KNX field layer, for example if more than one group address is used to control the same input (master control). Upon saving the NETx BMS Server loads the Link Definitions anew and executes them while remaining running. No restart is necessary. For more detailed information please refer to Section 4.3.1.6.

3.1.9.11. Send telegram...

Allows to send a KNX telegram (WRITE, READ or RESPONSE) to a given group address via a given interface.

3.1.9.12. Set cell value...

Allows to set the value of a given KNX group address on a given interface to a speciic value. This value is set in the BMS Server only and is not sent to the fieldbus.

3.1.9.13. Advanced Configuration

[Live] N-Mesh Routing Definitions
In this table the routing information is held. If in N-Mesh Configuration “Enable Routing” is enabled, these definitions are executed. Should the routing of all Server Items be enabled, this definition is obsolete and all of them are routed to the destination server. Upon saving the NETx BMS Server loads the N-Mesh Routing Definitions anew and executes them while remaining running. No restart is necessary.

[Live] Response Event Definitions
Here you can define events based on receiving KNX telegrams. New KNX telegrams will be created when these events occur. Upon saving the NETx BMS Server loads the Response Event Definitions anew and executes them while remaining running. No restart is necessary. For more detailed information please refer to Section 4.3.1.6.

[Live] Timer Event Definitions
Here you can define events based on a timer. New KNX telegrams will be created when these events occur. Upon saving the NETx BMS Server loads the Timer Event Definitions anew and executes them while remaining running. No restart is necessary. For more detailed information please refer to Section 4.3.1.6.

[Live] Cyclic Event Definitions
Here you can define events cyclically reoccurring. New KNX telegrams will be created when these events occur. Upon saving the NETx BMS Server loads the Cyclic Event Definitions anew and executes them while remaining running. No restart is necessary. For more detailed information please refer to Section 4.3.1.6.

3.1.10. BACnet

3.1.10.1. Start Explorer

Starts the BACnet Explorer to analyze the network and save the data to the configuration files (cf. Section 3.4.2). After each change in the definition files the NETx BMS Server has to be restarted to apply changes.

3.1.10.2. Device Definitions

Here you can edit the BACnet device definition table. After changing it it is recommended to restart the system. For any more detailed information about the single fields of the Device Definitions please refer to Section 4.3.3.2.

3.1.10.3. Object Definitions

Here you can edit the BACnet object definition table. After changing it it is recommended to restart the system. For any more detailed information about the single fields of the Object Definitions please refer to Section 4.3.3.3.

3.1.10.4. Object Mapping Definitions

Here you can edit the BACnet object mapping definition table. After changing it it is recommended to restart the system. For any more detailed information about the single fields of the Object Mapping Definitions please refer to Section 4.3.3.4.

3.1.10.5. Driver Configuration

This will open the connection configuration dialog for the BACnet interface. For detailed information about each parameter refer to Section 4.3.3.1. After acknowledging the content of the dialog in clicking “OK” it is recommended to restart the system. Any made changes will become effective then.

3.1.11. Modbus

3.1.11.1. Device definitions

Here you can define the Modbus devices. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the device definitions please refer to Section 4.3.2.1.

3.1.11.2. Datapoint definitions

Here you can define the Modbus datapoints. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the datapoint definitions please refer to Section 4.3.2.2.

3.1.11.3. Address mapping definitions

Here you can define the Modbus address mappings. After changing them it is recommended to restart the system. For any more detailed information about the single fields of the address mapping definitions please refer to Section (cf. Section 4.3.2.3).

3.1.12. JSON

The ABB JSON interface connects to gateways which communicate to different meters on their side again. Thus it is possible to communicate with them via HTTP. The system administrator gains a simple way to set up any meters in a clearly arranged overview.

3.1.12.1. Gateway Definitions

This table contains the definitions for the communication with the single JSON gateways such as ID or IP Address. For any detailed information about the settings refer to Section 4.3.4.1. After any change it is recommended to restart the system to have them come into effect.

3.1.12.2. Meter Definitions

In here the meters connected to a gateway are defined. Specifications like type or the superior gateway id are made there. Detailed information to each of the columns can be found in Section 4.3.4.2. It is recommended to restart the system after any changes made in this configuration table to have them come into effect.

3.1.12.3. Driver Configuration

This will open the connection configuration dialog for the JSON interface. For detailed information about each parameter refer to Section 4.3.4.3. After acknowledging the content of the dialog in clicking “OK” it is recommended to restart the system. Any made changes will become effective then.

3.1.13. SNMP

3.1.13.1. Device definition

Opens the device definition file to configure the SNMP devices which should be monitored.

3.1.13.2. Polling definition

The definition of polling items provide the monitoring of single parameter of a specific SNMP device.

3.1.13.3. User definition

The function opens the user definition file. The user definitions are used for the authentication process which is required for the SNMP Version 3.

3.1.13.4. Traps

Trap definition
The opened file provide the definition of trap messages which enable to react on specific events which are reported by the SNMP device.

Variable definition
This function opens the variable definition file which are used to split received trap messages into single data points of the NETx BMS Server.

Listener definition
The opened definition file enables to specify IP addresses and ports where the NETx BMS Serverlistens for incoming trap messages.

3.1.13.5. Driver definition

Opens the SNMP driver configuration file to define the global parameter for the XIO module.

3.1.14. Tools

3.1.14.1. KNX Telegram History Explorer...

Starts the *KNX Telegram History Explorer*, with which the logged telegrams can be analyzed. For more detailed information about the App please refer to Section 3.4.3.

3.1.14.2. Execute LUA script...

This function will open a script editor to execute immediately LUA script. The “Execute” button will run the script inside the editor while the “Cancel” button closes the window.

```
nxa.LogInfo("Hello World!")
```

The above example script will write a message “Hello World!” into the System Messages. For any further information about the LUA script engine in the NETx BMS Serverplease refer to Section 4.6.

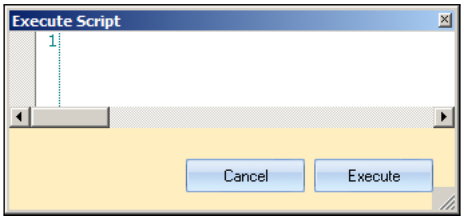


Figure 3.15.: Execute Script

3.1.14.3. Load XCommand...

Loads an XCommand manually into the NETx BMS Server cache to allow execution in the NETx BMS Server Studio. Generally, XCommands are loaded during NETx BMS Server startup. Use this operation to use the selected XCommand without the need of a server restart.

3.1.14.4. Run XCommand...

Runs an XCommand. In a dialog the user selects the XCommand to be executed and sets both required and optional inputs, outputs and parameters.

3.1.14.5. Options...

This entry will open the “Options...” menu to adjust the behavior of the NETx BMS Server Studio. The following parameters are available:

Parameter:	**Language of BMS Studio**
Scope:	ENUM; English, Deutsch
Default value:	English
Unit:	None
Description:	Choose the language of the NETx BMS Server Studio GUI.

Parameter:	**Code page**
Scope:	ENUM; West European Latin, Central and East European Latin, Cyrillic, Greek, Turkish, Hebrew, Arabic, Baltic, Thai
Default value:	West European Latin
Unit:	None
Description:	Choose the code page for font display of the NETx BMS Server Studio GUI.

3.1.15. Windows

The following list shows you the functionality of the menu commands only. For more detailed information about the single window sections please refer to Section 3.3.

3.1.15.1. Add Item Tree

Adds an Item Tree view with every click up to a count of four.

3.1.15.2. Project tree

Switches the “Project Tree” window on or off.

3.1.15.3. System Messages

Switches the “System Messages” window on or off.

3.1.15.4. Telegram Monitor

Switches the “Telegram Monitor” window on or off.

3.1.15.5. Cell monitor

Switches the “Cell monitor” window on or off.

3.1.15.6. Gateway manager

Switches the “Gateway Manager”-window on or off.

3.1.15.7. Search

Switches the “Search” window on or off.

3.1.15.8. Item properties

Switches the “Item properties” window on or off.

3.1.15.9. Graph

Switches the “Graph” window on or off.

3.1.15.10. Restore Positions

It restores the position of all windows to its default position.

3.1.15.11. Cascade

It positions the definition windows in the center area as cascade.

3.1.15.12. Vertical

It positions the definition windows in the center area vertically.

3.1.15.13. Horizontal

It positions the definition windows in the center area horizontally.

3.1.16. Info

3.1.16.1. Documentation...

Opens the documentation of the NETx BMS Server (requires a PDF reader).

3.1.16.2. License manager...

This menu entry will appear only when there is no valid hard lock key(USB stick) available. It opens the licensing App to register the server with the soft lock method. For any further information please refer to Section 3.4.4.

3.1.16.3. About BMS Server

This opens the “About” dialog of the application. Some general information like the exact NETx BMS Server Studio Version can be viewed here.

3.2. Toolbar

The tool bar grants the user direct access to the main tools and areas of the NETx BMS Server Studio.

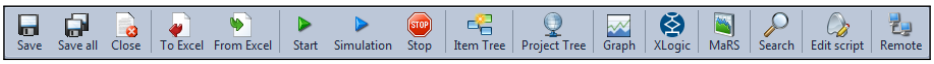


Figure 3.16.: Toolbar

3.2.1. Save

Saves the currently opened configuration table. The user is asked whether or not to restart the server. If yes, the NETx BMS Server will be restarted using the new table.

!Attention: Restarting the NETx BMS Server will disconnect all clients.

3.2.2. Save all

Saves all open definition tables. The user is asked whether or not to restart the server. If yes, the NETx BMS Server will be restarted using the new tables.

!Attention: Restarting the NETx BMS Server will disconnect all clients.

3.2.3. Close

Closes the currently opened configuration table.

3.2.4. To Excel

This exports the active definition table into a Microsoft©Excel file.

3.2.5. From Excel

The function imports the active definition table from a Microsoft©Excel file and overrides the current contents.

3.2.6. Start

Starts the NETx BMS Server in ONLINE mode in which all field layer communication is carried out physically.

3.2.7. Simulation

Starts the NETx BMS Server in simulation mode. No signals will be sent to the field layer. It is possible to set item values in the tree. Therewith a manual test of OPC Clients is possible. If the generation of random values is activated the server simulates some activity. This again is visible on the OPC Clients to test.

3.2.8. Shutdown

This will shut down the NETx BMS Server.

3.2.9. Item Tree

Adds an Item Tree view with every click up to a count of four.

3.2.10. Project tree

Switches the “Project Tree” window on or off.

3.2.11. Graph

Switches the “Graph” window on or off.

3.2.12. XLogic

Starts the XLogic editor for graphical logic programming.

3.2.13. MaRS

Starts the NETx MaRS Module (Metering and Reporting System).

3.2.14. Search

It enables or disables the search tool for Server Items.

3.2.15. Edit Script

This function starts the built in editor for LUA scripts. First a file open dialog will appear whereupon the selected file will be displayed in the editor.

3.2.16. Remote

The function starts the built in support tool which can be used to create a remote desktop connection for maintenance and support.

3.3. Window

3.3.1. Menu Bar

This area was already described in Section 3.1. For its exact location when the application windows are in their default position refer to number 2 in Figure 3.1.

3.3.2. Tool Bar

This area was already described in Section 3.2. For its exact location when the application windows are in their default position refer to number 3 in Figure 3.1.

3.3.3. Info Bar

The Info Bar shows a few important values of the server in a very compact way. The values are mostly KNX related. For its exact location when the application windows are in their default position refer to number 4 in Figure 3.1.

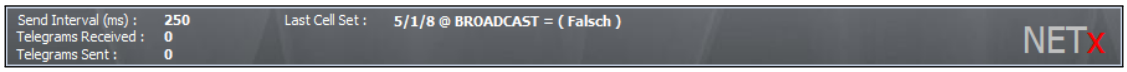


Figure 3.17.: Info Bar

Send Interval
Shows the current value (milliseconds) of the interval in which telegrams are sent to a single KNX gateway.

Last Cell Set
This displays address and the value of the last sent KNX telegram.

Telegrams Received
It shows the amount of the received KNX telegrams since the last initializing of the server.

Telegrams Sent
It shows the amount of sent telegrams since the last initializing of the server.

3.3.4. Project Tree

This area contains an overview about the NETx BMS Client definitions and the connected visualization projects. For its exact location when the application windows are in their default position refer to number 5 in Figure 3.1.

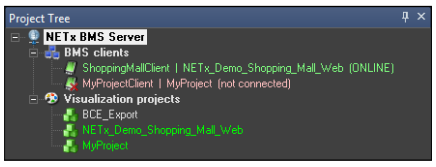


Figure 3.18.: Project Tree

BMS Clients
This part of the tree contains all NETx BMS Client definitions of the server workspace. The colors of the single entries have a meaning to them:

- red – An error occurred; the NETx BMS Client cannot be connected.
- rose – The NETx BMS Client is not connected; the connection is on standby.
- green – The NETx BMS Client is connected; the connection has been established.
- white – No life data is available; the server is stopped.

A right click to the “BMS Clients” node or one of the client definitions will open a context menu from which the same functions can be selected that can be found in the “Visualization” menu (cf. Section 3.1.5).

Visualization Projects
In here the NETx BMS Client projects of the current server workspace are listed. Like in the previous node, here too, the colors of the single entries have a meaning.

- green – A NETx BMS Client workspace is associated to the NETx BMS Client project.
- white – Only a NETx BMS Client project file is linked to the server. The project itself cannot be manipulated directly with the NETx BMS Client Editor.

When a NETx BMS Client workspace is associated to the NETx BMS Client project a double click opens the workspace in the NETx BMS Client Editor. A right click to the “Visualization Projects” node or one of the NETx BMS Client project will open a context menu from which the same functions can be selected that can be found in the menu bar / “Visualization” (cf. Section 3.1.5).

3.3.5. Gateways

This area is listing the defined KNX gateways. For its exact location when the application windows are in their default position refer to number 6 in Figure 3.1.

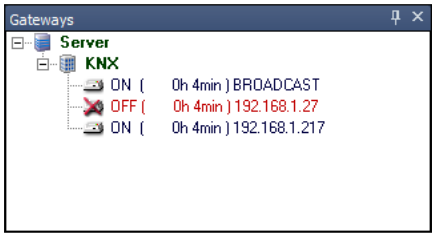


Figure 3.19.: Gateways

A red displayed gateway is not connected to the server. Double clicking the server opens the “Gateway Info” in the working area of the NETx BMS Server Studio. This can also be achieved in opening the context menu.

3.3.6. Cells

This area is listing datapoints to monitor them. For its exact location when the application windows are in their default position refer to number 7 in Figure 3.1.

To add a KNX datapoint to the monitoring table select the gateway in the drop down list and put in the group address. Then press the “Add” button. The datapoint will appear in the cell list. To delete the datapoint of the list the line of the table needs to be marked and the delete button to be pressed. The cell will be erased. Also from the item tree the right click command “Add to monitor” will insert the item to the cell monitor list.

3.3.7. Main Working Area

This area contains the main working space of the NETx BMS Server Studio. For its exact location when the

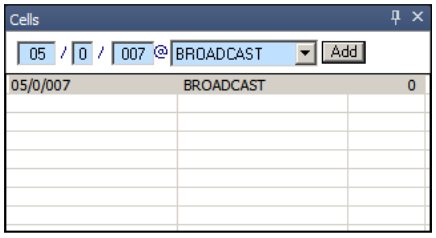


Figure 3.20.: Cells

application windows are in their default position refer to number 8 in Figure 3.1.

The Main Working Area can contain tabs with the following possible purposes:

- Item Tree – This displays the tree of all available Server Items.
- Search Datapoints – A search tab to find specified Server Items.
- Definition Tables – They can be displayed and edited here.
- Definition Files – A text editor is offered to manipulate them inside the studio.
- LUA Editor – A special editor with syntax highlighting is offered to create or manipulate LUA scripts.
- Log Viewer – A table will be displayed with the log entries.

3.3.7.1. Item Tree

The Item Tree displays all available Server Items of the current system. It displays these only if the NETx BMS Server is running either in online or simulation mode.

The values of the items are shown in the respective column and are continuously updated.

Server Items can be directly accessed. A right click to one will bring the context menu to display. The following three menu entries are available for all items:

- Show history in Graph – This brings up the “add” dialog for the graph (cf. Section 3.3.10).
- Copy Item path to clipboard – The full Server Item path will be copied to the clipboard.
- Show Properties – This shows the properties of the Server Item. If the “Properties” area is disabled it will be brought up again.

Other options like “Write Item Value. . . ” or “Set Item Value. . . ” will appear only on specific Server Items. Most of them bring masks up to actively change the value of the Server Item or to send a telegram directly to the fieldbus layer.

“Add to monitor” will insert the item into the cell monitor list.

3.3.7.2. Search Datapoints

This tab contains in its head a drop down list where it can be chosen wherein the search should take place. The text field next to it needs to contain the search string. The magnifier glass button on the right hand side starts the search. Alternatively the “return” key can be pressed.

The search string may contain regular expressions.

Once the result is displayed a context menu can be called from each item. It is possible to add the specified item to the Graph and copy the item path to the clipboard. A single click to a line will display the properties of the item in the according area. If the “Properties” area is disabled a double click to the line will call it to display.

3.3.7.3. Definition Tables

Tables such as “Task Definitions” or “Modbus Devices” are also opened in the Main Working Area. The context menu (right-click in a line) contains the following entries:

- Insert new definition – This inserts a new definition line below the active line.

- Insert new comment – This inserts a new comment line below the active line.
- Convert to comment/definition – This will convert the active line into a comment or definition depending on its previous state.
- Copy – Copy the selected line(s) to the system clipboard.
- Cut – Cuts the selected line(s) and keeps a copy in the system clipboard.
- Paste – Pastes contents of the clipboard in the current position.
- Delete – This will delete the active line.

Other options like “Send Telegram . . . ” or “Set Cell Value. . . ” will appear only in KNX Telegram Definitions. They bring masks up to actively change the value of the Server Item or to send a telegram directly to the fieldbus layer. “Add to monitor” will insert the item into the cell monitor list.

3.3.7.4. Definition Files

An editor will be opened to manipulate definitions like the “Router Config File”.

3.3.7.5. LUA Editor

The LUA Editor offers a lot more functionality than the one for the definition files. Next to syntax highlighting, row counting and printing it also provides a search and replace dialog. The LUA script engine can be reinitialized at any time while the NETx BMS Server is running.

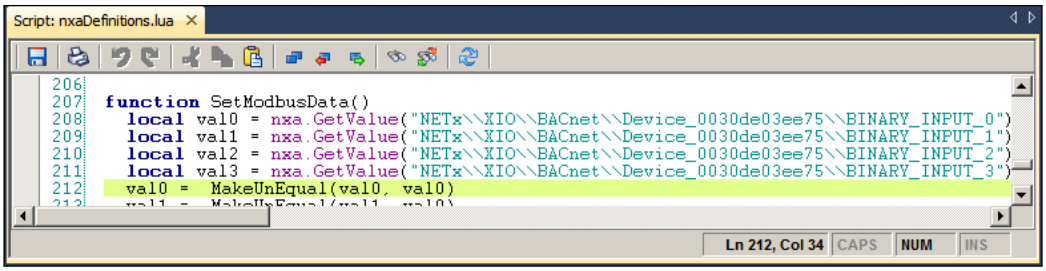


Figure 3.21.: LUA Editor

Below standing described are the symbols of the LUA Editor in order from left to right.

Save [Ctrl] + [S]
This will save the actual opened LUA file.

Print
The actual LUA file will be printed.

Undo [Ctrl] + [Z]
The last operation will be undone.

Redo [Ctrl] + [Y]
If an operation was undone it can be redone.

Cut [Ctrl] + [X]
The marked script text will be cut off the editor and copied to the clipboard.

Copy [Ctrl] + [C]
The marked script text will be copied to the clipboard.

Paste [Ctrl] + [V]
The last text copied to the clipboard will be inserted at cursor position.

Bookmark ON/OFF
Bookmarks can be set or erased on the active line.

Goto Prev Bookmark
This command will navigate forward/downward through the set bookmarks.

Goto Next Bookmark
This command will navigate backward/upward through the set bookmarks.

Find [Ctrl] + [F]
A dialog will appear where patterns can be entered to search in the actual file.

Find and Replace [Ctrl] + [H]
A dialog will appear where patterns can be entered to search in the actual file. These patterns can be replaced with a second set entered in the same dialog.

Reload and restart the Script Engine
This will load the actual saved state of the LUA files into the script engine and restart it.

! Initial functions will be executed at server start only. Restarting the LUA engine will not execute these functions again. If the engine needs to be reinitialized a server restart is necessary.

3.3.7.6. Log Viewer (System Log)

The System Log File in its viewer offers more navigation functionality than the definition tables.

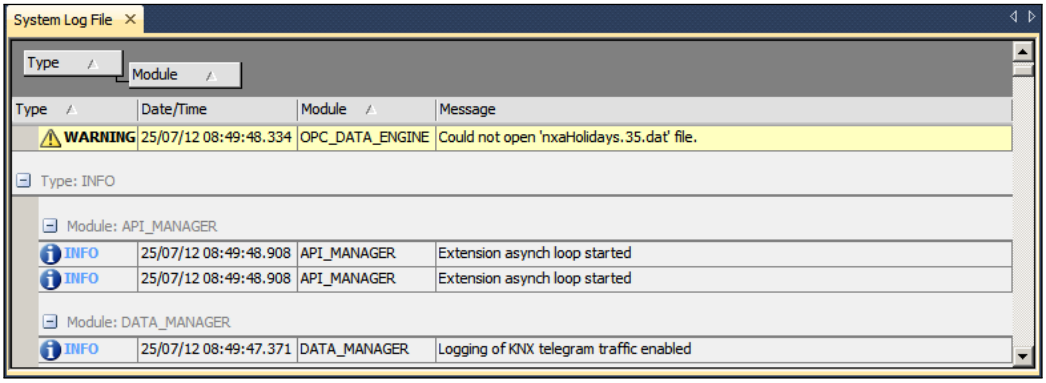
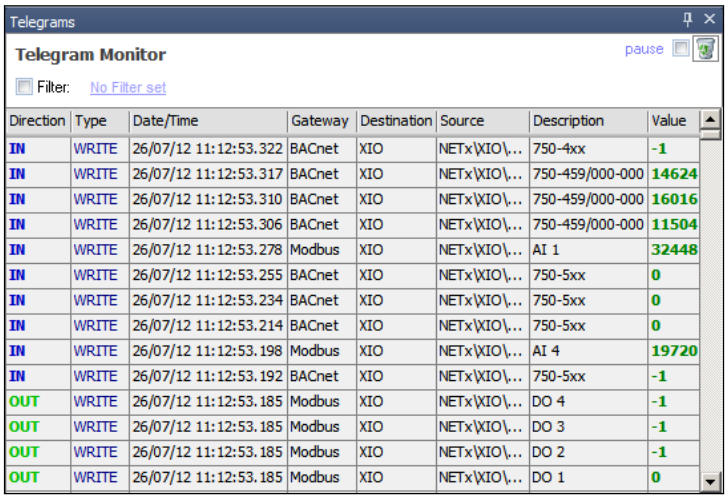


Figure 3.22.: Log Viewer

If a column header is dragged and dropped to the group field the header will be pulled off the table and the data will be grouped by it. If you hold the [Ctrl] key pressed while dragging the header, it will remain in the table. Multiple grouping is possible.

3.3.8. Telegrams

This area is listing all telegrams to monitor them. They can be filtered and paused to get a more detailed impression of them. For its exact location when the application windows are in their default position refer to number 9 in Figure 3.1.



Direction	Type	Date/Time	Gateway	Destination	Source	Description	Value
IN	WRITE	26/07/12 11:12:53.322	BACnet	XIO	NETx\XIO\...	750-4xx	-1
IN	WRITE	26/07/12 11:12:53.317	BACnet	XIO	NETx\XIO\...	750-459/000-000	14624
IN	WRITE	26/07/12 11:12:53.310	BACnet	XIO	NETx\XIO\...	750-459/000-000	16016
IN	WRITE	26/07/12 11:12:53.306	BACnet	XIO	NETx\XIO\...	750-459/000-000	11504
IN	WRITE	26/07/12 11:12:53.278	Modbus	XIO	NETx\XIO\...	AI 1	32448
IN	WRITE	26/07/12 11:12:53.255	BACnet	XIO	NETx\XIO\...	750-5xx	0
IN	WRITE	26/07/12 11:12:53.234	BACnet	XIO	NETx\XIO\...	750-5xx	0
IN	WRITE	26/07/12 11:12:53.214	BACnet	XIO	NETx\XIO\...	750-5xx	0
IN	WRITE	26/07/12 11:12:53.198	Modbus	XIO	NETx\XIO\...	AI 4	19720
IN	WRITE	26/07/12 11:12:53.192	BACnet	XIO	NETx\XIO\...	750-5xx	-1
OUT	WRITE	26/07/12 11:12:53.185	Modbus	XIO	NETx\XIO\...	DO 4	-1
OUT	WRITE	26/07/12 11:12:53.185	Modbus	XIO	NETx\XIO\...	DO 3	-1
OUT	WRITE	26/07/12 11:12:53.185	Modbus	XIO	NETx\XIO\...	DO 2	-1
OUT	WRITE	26/07/12 11:12:53.185	Modbus	XIO	NETx\XIO\...	DO 1	0

Figure 3.23.: Telegrams

The check box “pause” in the top right corner of this area sets the monitor on hold. The telegrams can be viewed now closely. The “dust bin” button next to the check box empties the content of the Telegram Monitor.

The check box “Filter” on the left hand side enables or disables the telegram filter. The filter settings can be called with the link next to it.

If the filter is set, only part of the telegrams sent or received are shown in the telegram monitor. This functionality will make analysis of the data traffic a lot easier. The filter consists of three sections. The module filter defines which XIO subsystem’s telegrams should be displayed. The telegram types filter defines which types of telegrams should be displayed. For KNX telegrams, the third section allows to filter a group address range and a specific device IP address. Settings of all three sections are combined with an AND operation. (c.f. Figure 3.24).

3.3.9. Properties

This area is listing all properties to a Server Item. For its exact location when the application windows are in their default position refer to number 10 in Figure 3.1.

In the head region of this area the description of the Server Item is displayed. Below it the full path of the Server Item can be found. From this place it can be copied to anywhere else.

The properties include “Item Value”, “Item Quality”, “Item Unit”, and many other fields. Properties fields are read-only.

3.3.10. Graph

This area contains the a graphical visualization for selected Server Items. For its exact location when the application windows are in their default position refer to number 11 in Figure 3.1.

The small check box on the graph surface enables or disables the auto refresh of the graph. It is possible to view the Server Items cyclically refreshed or static for a closer analysis. The refresh button reloads all datapoints anew. A day or week to view can be selected with the calendar button. The time span is selected from 00:00am to 12:00pm. The zoom button displays all actual datapoints in one graph screen. The next two buttons allow it to navigate in zoom history back and forth, where the two magnifier glasses zoom in and out. With the drop down list it is possible to switch between the three cursor modes; zoom, pan, and info.

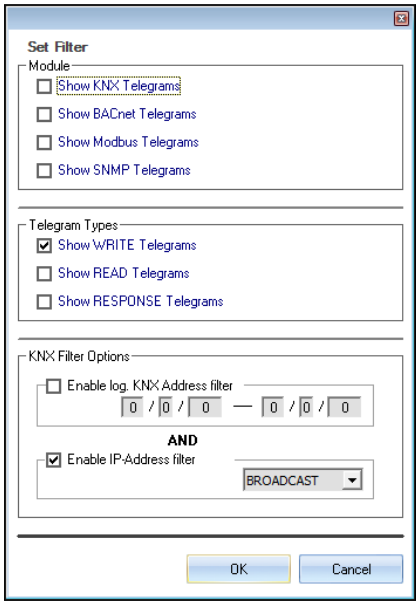


Figure 3.24.: Set Filter

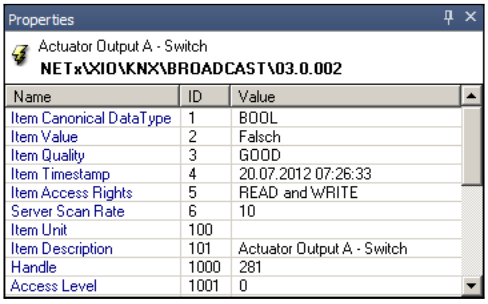


Figure 3.25.: Properties

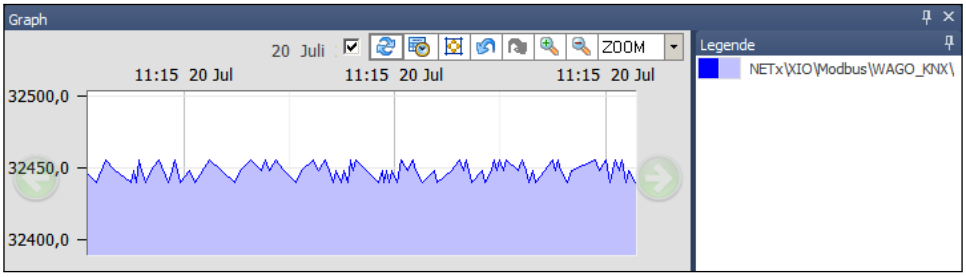


Figure 3.26.: Graph

On the right hand side the legend can be found. This area can be opened, fixed (with the pin needle) and dragged around like all other window areas as well.

Server Items can be added to the graph by dragging them from the item tree directly onto the graph area. A small dialog will appear where the user can make a few settings for displaying the item (cf. Figure 3.27).

All Server Items to be displayed on the graph need to be set “Historical”. Their values will be recorded in the database.

A double click onto the graph opens a dialog where the Server Items attached to the graph can be managed. Comment, Line color, Filled, Fill color, Line type, Line width, and Visibility can be adjusted here. The red cross on the right hand side erases a line from the Graph. All changes will be made only when the [Apply] button or the [OK] button is pressed. Otherwise all made changes will be discarded.

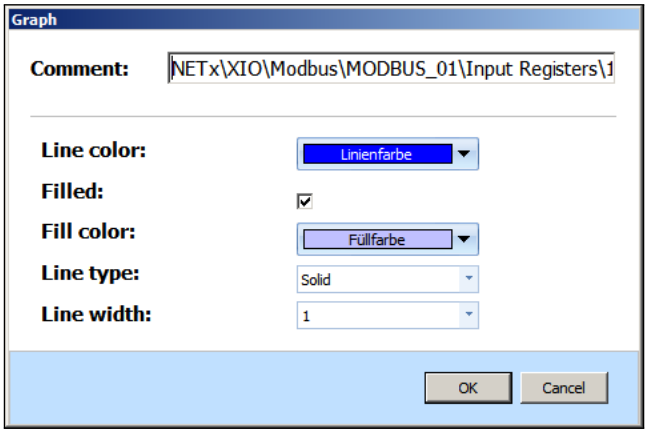


Figure 3.27.: Graph Add

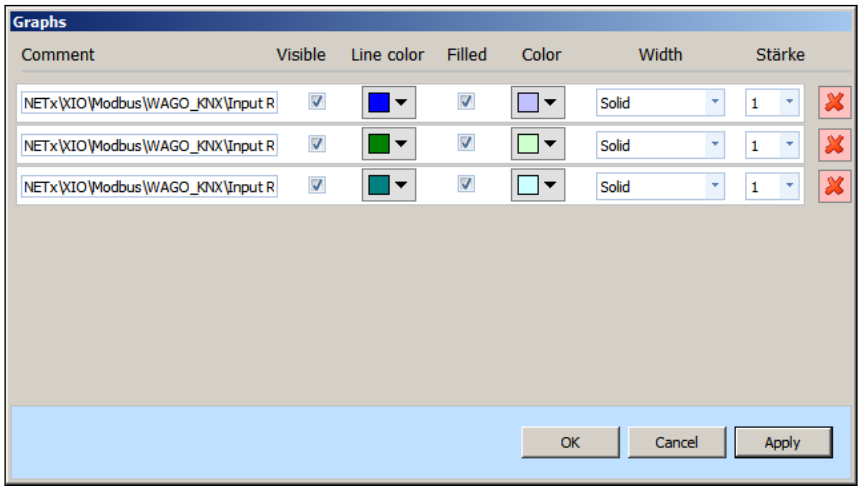


Figure 3.28.: Graph Edit

3.3.11. System Messages

In this area all the relevant events of the system are shown. For its exact location when the application windows are in their default position refer to number 12 in Figure 3.1.

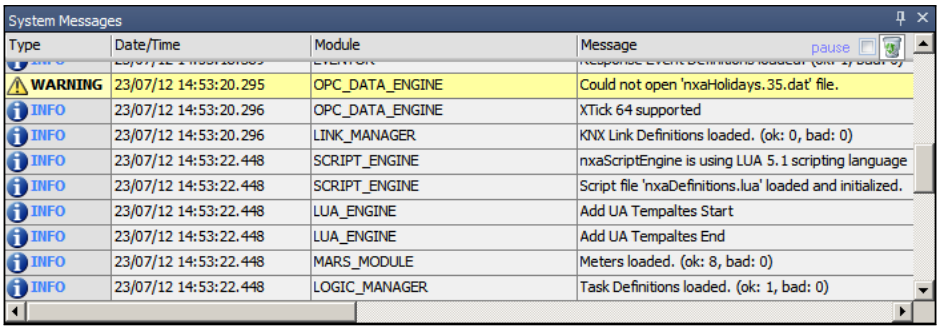


Figure 3.29.: System Messages

The background of any message shows which message type is displayed in this line:

- RED – Error messages.

- YELLOW – Warning messages.
- BLUE – Info messages.

The check box on the right hand side pauses the System Messages to update. A closer investigation of the actual displayed log entries is possible with it. The trash icon next to it empties the displayed entry list. All entries are stored in the “nxaOPCSystem.35.log” file for analysis at a later point of time. Emptying the displayed System Messages does not influence the content of the log file; its purpose concerns display only.

3.3.12. Statusbar

This area is next to the Info Bar the second information area. It shows server relevant data for a quick health overview. For its exact location when the application windows are in their default position refer to number 13 in Figure 3.1.

Status
This displays the current status of the server.

- Running
- Stopped

Start Date/Time
The last time when the server was started will be displayed here.

Mode
One of the following running modes completed by the name of the current workspace will be displayed in this field.

- ONLINE
- OFFLINE
- SIMULATION

N-Mesh Mode
This shows one of the following N-Mesh Modes. Additional to this it displays the activity of the server.
(e.g. “Backup Server (Passive)”)

- Stand Alone Server
- Main Server
- Backup Server

XDB
This field displays the database connectivity status of the NETx BMS Server Studio.

- ONLINE
- OFFLINE

3.4. Apps

3.4.1. NETxKNX ETS Converter 3.5

This App allows importing KNX Telegram Definitions, KNX Link Definitions, and KNX Group Aliases from ETS exported ESF files.

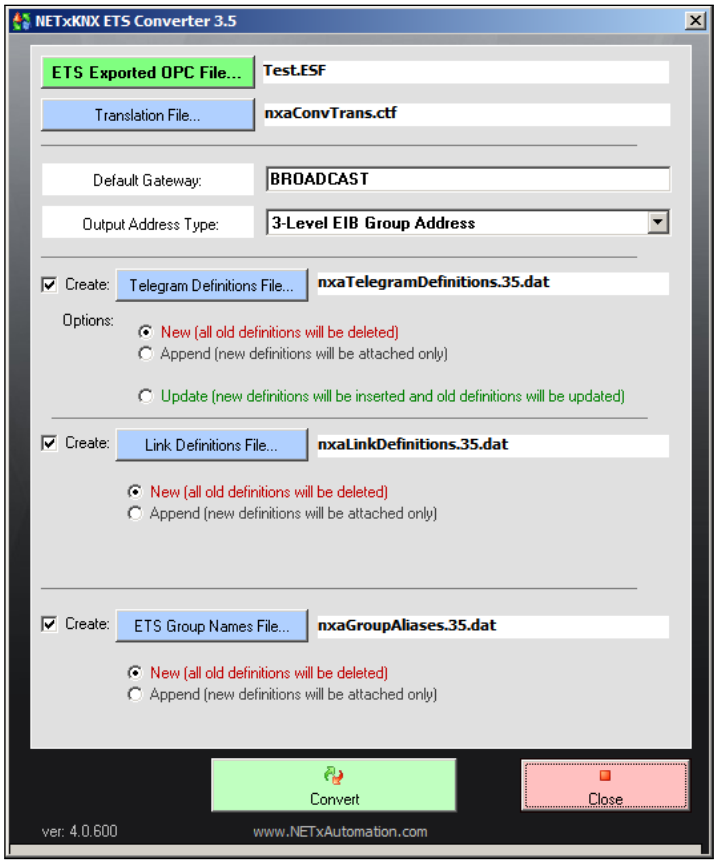


Figure 3.30.: NETxKNX ETS Converter 3.5

ETS Exported OPC File...
The button [ETS Exported OPC File...] opens a file dialog where the exported ESF file can be selected.

Translation File...
The button [Translation File...] opens a file dialog where the translation file can be selected. This file contains the data type definitions for the conversion. It is possible to maintain this file or select a customized one for an import.

Default Gateway
Enter the address for the gateway which will be associated with the group addresses in this text field. Imports for different gateways need to be done separately.

Output Address Type
Set the number of levels for the group addresses in here.

Telegram Definition File
Set the check box to import the telegram definitions. The button will open a file dialog where the output file can be selected. The option “New” will create a complete new file, while “Append” will cause the converter to add all group addresses which are not in the telegram definition yet. The option “Update” will add those group addresses, which are not in the telegram definition yet and in addition will update the existing ones if changes were made to them.

Link Definition File
Set the check box to import the link definitions. The button will open a file dialog where the output file can be selected. The option “New” will create a complete new file, while “Append” will cause the converter to add all links which are not in the link definition yet.

ETS Group Names File
Set the check box to import the alias definitions. The button will open a file dialog where the output file can be selected. The option “New” will create a complete new file, while “Append” will cause the converter to add all aliases which are not in the name file yet.

Convert
This will start the conversion process. And once confirmed the result will be written to the files.

Close
This will close the App.

3.4.2. BACnet Explorer

The App scans the network for BACnet devices and their datapoints (cf. Figure 2.7).

Result
The largest frame displays the scan result of the network. A click onto a device will check it and scan its datapoints. The resulting lines are checked by default. Unwanted datapoints need to be unchecked.

Scan
The frame in the lower left corner contains the scan log and a button to rescan the devices.

Export
This frame contains the options for exporting the BACnet data. “New” will in both cases “Export Devices” and “Export Objects” discard the former definition file and create a complete new one with the actual scanned data. “Append” will check for existing devices or datapoints (objects) and add only new ones to the existing definitions.

3.4.3. Telegram History Explorer

The Telegram History Explorer converts telegram history log files into a readable format.

File Configuration
The Source Telegram Log File defines from which log file the history should be read while the Output Telegram Text File will contain the result of the conversion.

Gateway IP Address
It is possible to run the conversion over the full log file or filter it by a specified IP address. The entry needs to be a valid IP address.

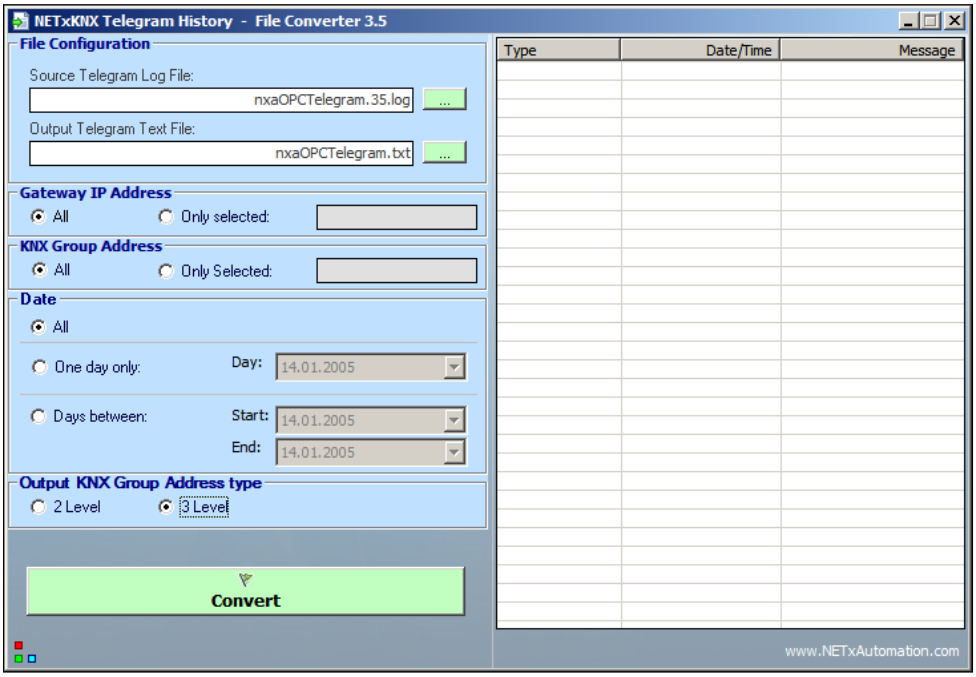


Figure 3.31.: History Explorer

KNX Group Address
It is possible to filter the conversion by a specified KNX group address. The entry needs to be a valid group address.

Date
In this option frame it is possible to set a time frame for which the conversion is to be done. “All” will convert the whole file while “One Day Only” will convert a specified day of interest and “Days Between” a specified time frame.

Output KNX Group Address Type
This setting determines the output format for the KNX Group Addresses. It is possible to convert the content of the log files into 2-Level or 3-Level KNX Group Addresses.

Convert
This will start the conversion.

Conversion Log
The table on the right hand side will contain info-, warning-, or error-messages when the conversion is started to inform the user about the process.

3.4.4. License Manager

The License Manager is needed only in case of a software license. The parts of this App are explained further down in Section A.2.2.1.

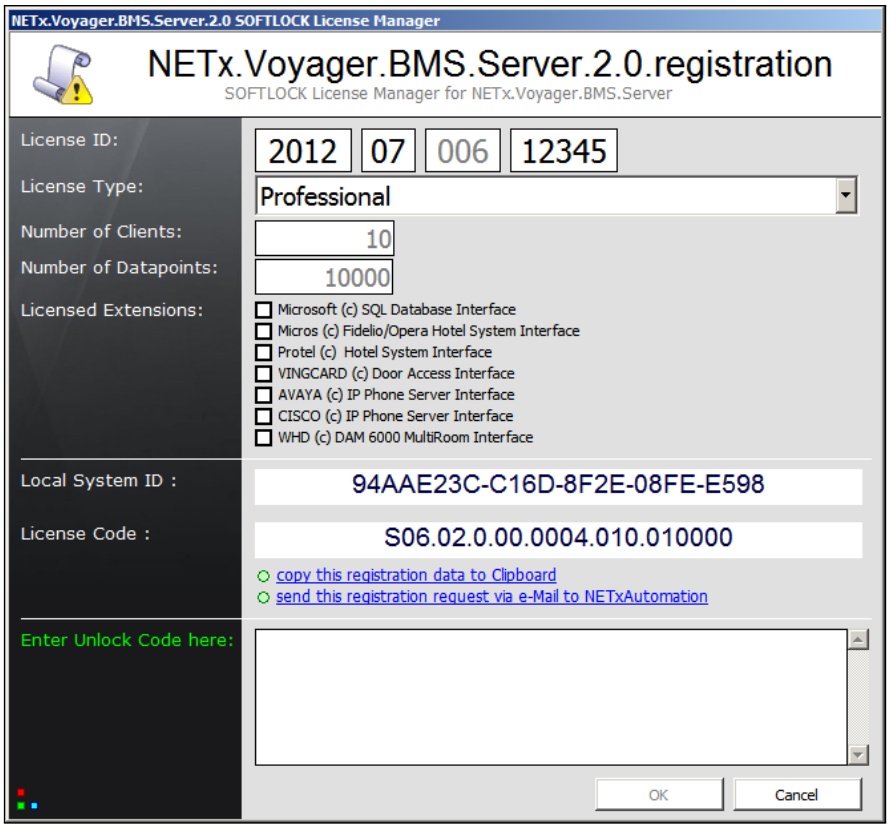


Figure 3.32.: License Manager

4. NETx BMS Server

The aim of the NETx BMS Server is to provide a uniform view of the data that is provided by the different sensors, actuators, and controllers located at the automation and field level within the building automation system. This abstract view of data is then represented as a hierarchically organized server item tree (cf. Section 1.3.2) that can be seen as a virtual model of the building automation systems. Using this virtual model, management clients can access the data and modify it.

4.1. The way how the server/service works

The server application itself is a Windows application that can be installed (i.e. registered) either as a “Windows Service” or as a “Windows COM Server”. If the NETx BMS Server is running as a Windows Service (which is the default setting after installation), the NETx BMS Server is automatically started during Windows start up. Therefore, no user has to log on to the system. If it is registered as a “Windows COM Server”, it is not started automatically. In this case, the server is only started when a client or the NETx BMS Server Studio itself connects to it.

Switching between these two kinds of execution modes can be done at anytime after installation. To change the execution mode, two Windows batch files are provided. These batch files are located within the installation path of the NETx BMS Server and have to be executed with administrator rights.

To register the NETx BMS Server as a Windows COM Server, the following command as to be executed:

```
Reg_Server.bat
```

To register it as a Windows Service again, the following batch file has to be invoked:

```
Reg_Service.bat
```

If the NETx BMS Server is installed as a Windows service, it is listed within the service list that is available within the Windows Control Panel as:

```
[NXA] NETx.VOYAGER.BMS.SERVER.2.0
```

4.2. General server configuration

All configuration data and parameters within a NETx BMS Server project are stored within a *workspace*. A workspace consists of various configuration files that are read during the start up procedure of the server. These files are all stored within the following directory:

```
<Install Directory>\NETxAutomation\NETx.Voyager.Server.2.0.UD\Workspaces
```

Within this directory, each NETx BMS Server project and each workspace has its own sub directory where all corresponding configuration and definition files are stored. The name of this directory is identical to the name of the workspace:

```
<Install Directory>\NETxAutomation\NETx.Voyager.Server.2.0.UD\Workspaces\<Workspacename>
```

Each workspace directory has the several sub directories that are home for the different workspace files.

Configuration data files contain general configuration parameters that change the behavior of the server functionality (e.g. IP address of the used network interface, . . .). These configuration data files are stored in:

```
<WorkspaceDirectory>\ConfigFiles
```

Datapoint configuration files are used to specify the different datapoints from the field level that shall be accessed by the server. The configuration of the field devices that handle these datapoints is also defined here. The corresponding configuration files are stored in:

```
<WorkspaceDirectory>\DataFiles
```

To add additional control functionality to the server, events can be defined. The required configuration files for the events are located within the following directory:

```
<WorkspaceDirectory>\EventFiles
```

Logging information that is generated by the server during runtime is located in:

```
<WorkspaceDirectory>\LogFiles
```

NETx BMS Client projects are stored within container files which use the file extension ".vxf". These NETx BMS Client files are stored within the following directory:

```
<WorkspaceDirectory>\ProjectFiles
```

In cases where the provided event definition mechanisms are not sufficient enough to add more control functionality, LUA scripts can be implemented and executed within the server's environment. These LUA scripts are located in:

```
<WorkspaceDirectory>\ScriptFiles
```

4.2.1. VNET Server Configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\nxaVoyagerServer.10.cfg
```

Within this configuration file, the VNET server that provides an interface to clients (e.g. BMS Clients, NETx Voyager, ...) is configured. Each parameter is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

Parameter:	**VNET.NetworkCardIPAddress**
Scope:	IP Address
Default value:	IP of default network interface
Unit:	None
Description:	It defines the IP address which is used for opening the VNET server socket. This IP address must be set within the configuration of the VNET clients. If no value is specified, the IP address of the system's default network interface is chosen.
Parameter:	**VNET.NetworkPortNumber**
Scope:	TCP Port
Default value:	4530
Unit:	None
Description:	It defines the TCP port which is used for opening the VNET server socket. This port number must be set within the configuration of the VNET clients.
Parameter:	**VNET.WebServerPort**
Scope:	TCP Port
Default value:	80
Unit:	None
Description:	It defines the port number which is used by the internal Web server of the NETx BMS Server. This port number must be used by the NETx BMS Clients in order to get access to the Web-based visualization.
Parameter:	**VNET.ClientTimeout**
Scope:	Duration
Default value:	300000
Unit:	milliseconds
Description:	This parameter defines the timeout of the VNET connections.

4.2.2. System Configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\nxaOPCSystem.35.cfg
```

Within this configuration, general system settings are specified. Each parameter is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

The parameters which can be defined in this file are divided into the following three groups:

- OPC
- KNX
- SYSTEM

4.2.2.1. OPC-Parameters

Parameter:	OPC.GroupAddressType
Scope:	2level, 3level
Default value:	3level
Unit:	None
Description:	It defines the format that is used to map KNX group addresses to server item IDs. 2Level – uses the two-level KNX group address format 3Level – uses the three-level KNX group address format

!Attention: In contrast to defined parameter within the studio, this has a crucial influence on the structure of all server item IDs.

The following examples represent the same KNX datapoint:

```
OPC.GroupAddressType: 2Level -> ITEM ID = "NETx\XIO\KNX\192.168.1.1\00/0513"
OPC.GroupAddressType: 3Level -> ITEM ID = "NETx\XIO\KNX\192.168.1.1\00/2/001"
```

Parameter:	OPC.PrefixedItemID
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	It defines whether the leading “0”-characters shall be added to server item IDs. ON – defines that leading “0”characters are added to the item IDs of KNX group addresses OFF – defines that the item IDs of KNX group addresses do not contain leading “0”-characters

!Attention: This parameter has crucial influence on the structure of all server item IDs.

The following examples represent the same KNX datapoint:

```
OPC.PrefixedItemID: ON -> ITEM ID = "NETx\XIO\KNX\192.168.1.1\00/2/001"
OPC.PrefixedItemID: OFF -> ITEM ID = "NETx\XIO\KNX\192.168.1.1\0/2/1"
```

Parameter:	OPC.AsyncTimeout
Scope:	1 – 60
Default value:	5
Unit:	seconds
Description:	It defines the timeout value for asynchronous OPC queries. If no answer to an OPC query is received within this time interval, the quality of all queried datapoints will be set to UNCERTAIN.

Parameter:	OPC.AsyncReadFromDevice
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It specifies whether the response data of the asynchronous OPC value read queries (“READ”) shall be read from the server’s data base, or whether it shall directly be retrieved from the field device. ON – the field devices are queried OFF – the values are read from the server’s data base

Parameter:	**OPC.AsyncRefreshFromDevice**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It specifies whether the response data of the asynchronous OPC value refresh queries (“REFRESH”) shall be read from the server’s data base, or whether it shall directly be retrieved from the field device. ON – the field devices are queried OFF – the values are read from the server’s data base

Parameter:	**OPC.ShowETSStructure**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It specifies whether to show an ETS like structure within the server item tree ON – shows the ETS like structure OFF – shows the NXA like structure

Parameter:	**OPC.QualityForValueNotSet**
Scope:	UNCERTAIN, BAD, NOT_CONNECTED
Default value:	UNCERTAIN
Unit:	None
Description:	It specifies which OPC Quality value shall be used for item values that are not yet set.

Parameter:	**OPC.Delimiter**
Scope:	String (” not allowed)
Default value:	\
Unit:	None
Description:	It specifies which delimiter shall be used for the item IDs within the server item tree.

! Following OPC parameters are available only for compatibility reasons. In future versions they may vanish.

Parameter:	**OPC.AsyncWriteConfirmation**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It specifies whether the response data of the asynchronous OPC value write queries (“WRITE”) shall be read from the server’s data base, or whether it shall directly be retrieved from the field device. ON – the field devices are queried OFF – the values are read from the server’s data base

!Attention: If the parameter is set on “ON”, each OPC “WRITE” command will result in sending up telegrams to the field network. In the case of KNX, up to three KNX telegrams are sent.

Parameter:	**OPC.RefreshOnEqualValue**
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	It specifies whether OPC “OnDataChange” events are generated by incoming values from the field level even if the values are unchanged. ON – the “OnDataChange” of events will be generated OFF – the “OnDataChange” of events will be generated only with changes of value

Example:

If the server receives two KNX telegrams with same value and same address (IP + KNX group address) and the option “OPC.RefreshOnEqualValue” is set to “OFF”, the OPC “OnDataChange” event is generated only for the first telegram, since the second telegram does not cause a change of value of the datapoint. If this option is set on “ON”, then two OPC “OnDataChange” events are generated.

!Attention: Use the following option only with caution!

Parameter:	OPC.AllValuesValid
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	If set to "'ON"', server items always have the quality "'GOOD"', even if their quality is "'BAD"' or "'UNCERTAIN"'. This option can be used for limited OPC clients that are not aware of OPC qualities other than "'GOOD"'.

4.2.2.2. KNX-Parameters

Parameter:	KNX.ServerPhysicalAddress
Scope:	0.0.0 – 15.15.255
Default value:	15.15.255
Unit:	None
Description:	It specifies the physical KNX address of the NETx BMS Server. This address is used by the outgoing KNX telegrams.

Parameter:	KNX.Timeout
Scope:	2 – 60
Default value:	10
Unit:	seconds
Description:	It specifies the timeout value for KNX "READ" request. This value is also used for read queries during the initialization phase of the system.

Parameter:	KNX.PhysicalDeviceTimeout
Scope:	1 – 600
Default value:	30
Unit:	seconds
Description:	It specifies the timeout value for the telegrams that are sent to check the availability of KNX devices. This value is used by device manager subsystem.

Parameter:	KNX.CyclicEventStartDelay
Scope:	10 – 600
Default value:	30
Unit:	seconds
Description:	It specifies the start delay for cyclic Events. This value is used by event engine of the server.

Parameter:	KNX.CyclicEventTelegramGap
Scope:	100 – 3000
Default value:	250
Unit:	milliseconds
Description:	It specifies the time gap between cyclic events (in milliseconds). This value is used by event engine of the server.

Parameter:	KNX.SetLinkOnReceive
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	The "link manager" the subsystem of the Server provides the opportunity to handle linked KNX group addresses that are defined within the ETS. This parameter activates the handling of these linked group addresses. ON – sets the linked KNX group addresses when corresponding KNX group telegram is received by the server OFF – not activated

Parameter:	**KNX.SetLinkOnSend**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	The “link manager” the subsystem of the BMS Server makes possible the managing of linked KNX group addresses. This parameter specifies whether the mechanism is to be released by sending KNX telegrams. ON – sets the linked KNX group addresses when sending KNX telegrams OFF – not activated

Parameter:	**KNX.ShowUndefinedTelegrams**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	If this parameter is set, received KNX telegrams that have an unknown KNX group address (i.e. a group address that is not defined within the server) will be shown in the message window of the NETx BMS Server Studio. ON – shows undefined KNX telegrams OFF – not activated

! Following KNX parameters are available only for compatibility reasons. In future versions they may vanish.

Parameter:	**KNX.RefreshAllValuesOnConnect**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	If a KNXnet/IP router is reconnecting to the server, the server can poll the current value of all datapoints that are assigned to this router. ON – the KNX datapoints are polled after a reconnect OFF – not activated

Parameter:	**KNX.CyclicRefreshAllValues**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It is possible that the system automatically updates the internal data base in defined time intervals. The updates are done by sending cyclic KNX “READ” telegrams ON – automatic updating is activated OFF – not activated

!Attention: Note that activating cyclic refreshing of KNX datapoints may cause high amount of network traffic that may stress the connected KNX lines.

Parameter:	**KNX.CyclicRefreshAllValuesInterval**
Scope:	5 – 86400
Default value:	10
Unit:	seconds
Description:	It specifies the time interval that is used for the cyclic refresh mechanism. Note that all defined KNX datapoints are queried during a refresh. Therefore, this interval has to be chosen depending on the amount of KNX datapoints within the project. If the value is too small, the previous refresh process may not be finish and thus it will overlap with the next.

Example:

10 datapoints for gateway 192.168.1.1 are defined. The global system send interval is set to 1 second (“send interval” can only be set in the NETx BMS Server Studio).

In this case, an update process takes 10 seconds. If the “KNX.CyclicRefreshAllValuesInterval” parameter is set to 5 seconds, the next update process already starts after 5 seconds and tries to send read telegrams. Since the telegrams of the first refresh process are still under transmission, the request of the second process cannot be sent and so these requests will remain in the queue of the server. To solve this issue, the “KNX.CyclicRefreshAllValues TelegramGap” parameter has to be set accordingly.

Parameter:	KNX.CyclicRefreshAllValuesStartDelay
Scope:	10 – 600
Default value:	60
Unit:	seconds
Description:	It specifies the delay of the first query process after the initialization of the server.

Parameter:	KNX.CyclicRefreshAllValuesTelegramGap
Scope:	100 – 3000
Default value:	250
Unit:	milliseconds
Description:	It specifies the time interval between two read request during a refresh process. The value of this interval must be greater than the value of the global system send interval.

Parameter:	KNX.ResetOnDisconnect
Scope:	On, Off
Default value:	On
Unit:	None
Description:	If set to “On”, the quality of KNX datapoints will be set to “UNCERTAIN” if the corresponding KNXnet/IP gateway is offline. If this parameter is set to “Off”, the KNX datapoints will remain “GOOD” even if the gateway is offline.

Parameter:	KNX.ReadIntervalFactor
Scope:	1 – 10
Default value:	0
Unit:	None
Description:	Using this parameter, the interval between two KNX read requests can be extended. The interval between two KNX read requests is the specified KNX sending value multiplied with this factor. A missing value or the value "‘0"’ means that the specified KNX sending interval is used.

4.2.2.3. SYSTEM-Parameters

Parameter:	SYS.Historizing
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	Enables or disables the storing of historical values into the SQL database.

Parameter:	SYS.BackupHistorizing
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	If enabled, the backup server also stores historical values in the historical database even if the backup server is inactive.

Parameter:	SYS.UseTelegramDataFile
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	It specifies whether a virtual image of the plant (i.e. the values of all defined datapoints) is written to disk.

Parameter:	SYS.SendVirtualTelegrams
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Enables or disables the sending of telegrams for KNX main group addresses above 15.

Parameter:	**SYS.MaxSizeOfTelegramLogFile**
Scope:	1 – 1000
Default value:	10
Unit:	Megabyte
Description:	It specifies the maximum size of the telegram log file.

Parameter:	**SYS.EnableDeviceManager**
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	This parameter enables the physical KNX device management. ON – the physical device management is enabled OFF – the physical device management is disabled

Parameter:	**SYS.GenerateRandomValues**
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	If the server is running in simulation mode and this parameter is set to “ON”, the server generates random values for all server items. ON – the server generates random values if the server is running OFF – The server does not generate random values

Parameter:	**SYS.EnableOperationTime**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	The server will calculate the operation time in seconds of KNX datapoints that are of type EIS1. ON – the operation time calculation is enabled OFF – the operation time calculation is disabled

Parameter:	**SYS.GeoLatitude**
Scope:	0 – 360
Default value:	0
Unit:	degrees
Description:	This parameter specifies the geographic latitude in degrees (+ is used for north, - for south)

Parameter:	**SYS.GeoLongitude**
Scope:	0 – 360
Default value:	0
Unit:	degrees
Description:	This parameter specifies the geographic longitude in degrees (+ is used for east, - for west)

Parameter:	**SYS.GeoElevation**
Scope:	0 – 10000
Default value:	0
Unit:	meters
Description:	It specifies the geographic elevation in meters.

! Following SYSTEM parameters are available only for compatibility reasons. In future versions they may vanish.

Parameter:	**SYS.GatewayConnectionTimeout**
Scope:	10 – 10000
Default value:	500
Unit:	milliseconds
Description:	It specifies the timeout value for availability check of the connected KNXnet/IP routers.

Parameter:	SYS.GatewayConnectionTimeoutAttemptCount
Scope:	1 – 10
Default value:	3
Unit:	None
Description:	It specifies the number of call attempts for the availability check of the connection of the connected KNXnet/IP routers in the case of an interruption.

Parameter:	SYS.CodePage
Scope:	
Default value:	1250
Unit:	None
Description:	Defines the code page that is used within the NETx BMS Server.

4.2.2.4. XDB-Parameters

The following parameters are important if a history of values or Metering data is recorded.

Parameter:	XDB.UseDSN
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It enables or disables the use of system Data Source Name (DSN) as a source of connection parameters. ON – The database module for historical data will use system DSN as connection parameter OFF – The database module for historical data will not use system DSN as connection parameter

Parameter:	XDB.UseIntegratedSecurity
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Use the integrated security mechanism of the operating system to connect to the database or use “user-name” and “password” instead.

Parameter:	XDB.UserID
Scope:	String
Default value:	None
Unit:	None
Description:	If the integrated security mechanism is not used, the specified user will be used to connect to the database.

Parameter:	XDB.Password
Scope:	String
Default value:	None
Unit:	None
Description:	If the integrated security mechanism is not used, the specified password will be used to connect to the database.

Parameter:	XDB.Hostname
Scope:	String[,Long 0-65535]
Default value:	None
Unit:	None
Description:	This is the host name where the database is running. It is possible to set it to “.” or “127.0.0.1” for connecting to the local machine. It is also possible to put in the DNS or the IP address of the data base computer. Optionally, the port number of the database instance can additionally be provided separated by a comma.

Parameter:	XDB.Instance
Scope:	String
Default value:	NETX_SERVER_SQL
Unit:	None
Description:	Instance name of the used database

Parameter:	XDB.DataSource
Scope:	String
Default value:	NETX_SERVER_DSN
Unit:	None
Description:	Data source name if XDB.UseDSN is set to “ON”

Parameter:	XDB.ConnectionTimeout
Scope:	Long 0-3600
Default value:	300
Unit:	seconds
Description:	Connection timeout

Parameter:	XDB.CommandTimeout
Scope:	Long 0-3600
Default value:	60
Unit:	seconds
Description:	Command timeout

Parameter:	XDB.MaxDataAge
Scope:	Long 0-1000
Default value:	0
Unit:	days
Description:	Maximum amount of days after which the historical data is deleted. 0 means that data is never deleted.

4.2.2.5. EMAIL-Parameters

The following parameters are important if an email system is established in XCON.

Parameter:	EMAIL.SmtpHost
Scope:	String
Default value:	None
Unit:	None
Description:	Specifies the IP address of the SMTP server.

Parameter:	EMAIL.SmtpPortNumber
Scope:	Long 1-65535
Default value:	0
Unit:	None
Description:	Specifies the port number of the SMTP server

Parameter:	EMAIL.SmtpUseAuth
Scope:	ON,OFF
Default value:	OFF
Unit:	None
Description:	Specifies whether to use SMTP authentication

Parameter:	EMAIL.SmtpUserName
Scope:	String
Default value:	None
Unit:	None
Description:	Specifies the SMTP user name

Parameter:	EMAIL.SmtpPassword
Scope:	String
Default value:	None
Unit:	None
Description:	Specifies the SMTP user password

Parameter:	EMAIL.SmtpEncryption
Scope:	String
Default value:	None
Unit:	None
Description:	Specifies whether to use SMTP encryption. Possible values NONE, TLS, or SSL

Parameter:	EMAIL.FromAddress
Scope:	String
Default value:	None
Unit:	None
Description:	E mail address that is used to send an e mail

Parameter:	EMAIL.AdminMailingList
Scope:	String
Default value:	None
Unit:	None
Description:	E mail address(es) that is used to send an admin e mail (e.g. “email1@test.org;email2@test.org”)

Parameter:	EMAIL.SystemMailingList
Scope:	String
Default value:	None
Unit:	None
Description:	E mail address(es) that is used to send a system e mail(e.g. “email1@test.org;email2@test.org”)

Parameter:	EMAIL.UserMailingList
Scope:	String
Default value:	None
Unit:	None
Description:	E mail address(es) that is used to send a user e mail(e.g. “email1@test.org;email2@test.org”)

4.2.3. N-Mesh Subsystem Config File

Location:

```
<WorkspaceDirectory>\ConfigFiles\nxaNMesh.35.cfg
```

Within this configuration, the N-Mesh subsystem is configured. The N-Mesh subsystem is used to set up a main/backup server environment as well as for defining server clustering. Main and backup server must have the same basic workspace configuration (e.g. datapoint definitions, device definitions, . . .). This is also true for servers that are part of a cluster environment. N-Mesh only works within environment that use the same release version.

Each parameter within the N-MESH configuration file is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

Parameter:	NMESH.UseRedundancy
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It enables redundancy feature of the server. This parameter has to be set to “ON” for both main and backup server as well as for all servers within the cluster environment. ON – Enables redundancy feature OFF – Disables the redundancy feature

Parameter:	NMESH.EnableSynchronization
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	This parameter enables the synchronization between the main and backup Server as well as between the server within a cluster environment. Enabling this feature, a server that is in standby mode will get all data changes from the active server.

Parameter:	NMESH.MainServerIPAddress
Scope:	IP Address
Default value:	192.128.1.1
Unit:	None
Description:	It defines the IP address of main server.

Parameter:	NMESH.BackupServerIPAddress
Scope:	IP Address
Default value:	192.128.1.2
Unit:	None
Description:	It defines the IP address of backup server.

Parameter:	NMESH.NetworkCardIPAddress
Scope:	IP Address
Default value:	192.128.1.1
Unit:	None
Description:	It defines the IP address of the network interface which is used to communicate with other N-Mesh servers. This parameter has to be individually within each server of the N-MESH environment.

Parameter:	NMESH.NetworkPortNumber
Scope:	0 – 65355
Default value:	20556
Unit:	None
Description:	It defines the port number which is used to communicate with other N-Mesh servers.

Parameter:	NMESH.RefuseClientsOffline
Scope:	ON,OFF
Default value:	OFF
Unit:	None
Description:	If set to “ON”, clients will be disconnected if server is in inactive mode and the client name contains the string “NMESH.BackupClientKeyword”.

Parameter:	NMESH.BackupClientKeyword
Scope:	String
Default value:	.BACKUP
Unit:	None
Description:	If “NMESH.RefuseClientsOffline” is set to “ON”, clients with a client name that contains the specified string are disconnected.

Parameter:	NMESH.StartDelay
Scope:	0 – 60
Default value:	10
Unit:	seconds
Description:	It defines the initialization delay of the N-MESH system after server start up.

Parameter:	NMESH.ConnectionTimeout
Scope:	10 – 10000
Default value:	500
Unit:	milliseconds
Description:	It defines the time out interval of the connection to the main server. If this time out interval elapses, the backup server decides that the main server is down. Afterwards, the backup server becomes active and takes over the control to ensure the availability of the system.

Parameter:	NMESH.EnableRouting
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It enables the N-Mesh routing of server items defined in N-Mesh routing definition file.

Parameter:	NMESH.Node
Scope:	IP Address
Default value:	None
Unit:	None
Description:	This parameter is used to define the structure of the cluster environment. If it is set, all update information is sent to the listed nodes. All N-Mesh nodes need the same datapoint definitions to store the receiving information, otherwise it is neglected. Note that NMESH nodes must also have an appropriate “NMESH.Node” entry where corresponding IP addresses of the sending nodes.

4.3. XIO Interfaces

The XIO interfaces of the NETx BMS Server are responsible for providing the access to the devices and datapoints that are located at the field and automation level of the building automation system. To achieve this, they implement the protocol stacks of the used network technologies. Within the current version of the NETx BMS Server, the following XIO interfaces are supported:

- KNX (cf. Section 4.3.1)
- Modbus (cf. Section 4.3.2)
- BACnet (cf. Section 4.3.3)

The remainder of this section describes these different XIO interfaces.

4.3.1. KNX XIO Interface

The KNX interface is used to integrate KNX data (e.g. KNX group objects, KNX devices) into the NETx BMS Server. The configuration files of the KNX interface are described in the remainder of this section. All certified KNX gateways that implement the KNXnet/IP tunneling protocol are supported. Also ABB IG/S 1.1 and b.a.b.-tec eibNode can be used as KNX gateways.

! Part of the configuration KNX data can directly be imported from ETS using the NETxKNXConvertETS App (cf. Section 3.4.1).

4.3.1.1. KNX configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\nxaOPCRouter.35.cfg
```

Within this configuration files, general configuration parameters of the KNX interface are set. Each parameter within the N-MESH configuration file is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

The following parameters are only relevant for ABB IG/S 1.1 and eibNode gateways:

Parameter:	UDP.ReceiveOwnTelegrams
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	This parameter specifies whether KNX telegrams that have the local IP address as source address are accepted by the server. ON – telegrams with local IP address are accepted OFF – local telegrams are ignored

Parameter:	UDP.ReceiveBroadcastTelegrams
Scope:	ON, OFF
Default value:	ON
Unit:	None
Description:	It specifies whether KNX telegrams from the “BROADCAST” gateway are accepted by the server. ON – “BROADCAST” telegrams are accepted OFF – “BROADCAST” telegrams are dropped

! If a KNX group address is only assigned to the “BROADCAST” router, all telegrams that use this KNX groups address are interpreted as “BROADCAST” telegrams – even if they are received from a KNXnet/IP router that is defined within the server.
If the KNX group address is assigned to another router (e.g. “192.168.1.2”) and a telegram with this KNX address is received from this router, the telegram will be interpreted as “192.168.1.2” telegram.

Parameter:	UDP.SendBroadcastAsMultipleUnicast
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	It defines whether the “BROADCAST” telegrams are sent as IP broadcast packets or as multiple unicast packets to each single router.

Parameter:	UDP.NetworkCardIPAddress
Scope:	IP address
Default value:	IP address of the default network interface
Unit:	None
Description:	It specifies the IP address of the network interface which is used to communicate with ABB IG/S gateways. If no IP address is specified, the IP address of the system’s default network interface is used.

The following parameters are only relevant for ABB IG/S 1.1 gateways:

Parameter:	IGS.ReceiveMulticastAddress
Scope:	IP address
Default value:	239.192.39.238
Unit:	None
Description:	It defines the multicast group (multicast IP address) that is used to communicate with ABB IG/S gateways.

The following parameters are only relevant for eibNode gateways:

Parameter:	eibNode.SenderNetID
Scope:	0 – 255
Default value:	0
Unit:	None
Description:	It specifies the eibNode NetID for outgoing telegrams.

Parameter:	eibNode.ReceiverNetIDFilter
Scope:	Comma separated list of values that are in the range of 0 – 255 (e.g. 0,15,36,223)
Default value:	<empty>
Unit:	None
Description:	It specifies the NetIDs of telegrams which are accepted by the server. If this parameter is left empty, all eibNode telegrams are accepted. For example, if “eibNode.ReceiverNetIDFilter” is set to “0,2,3,7”, only the telegrams with a NetID of 0,2,3, or 7 are accepted.

The following parameters are only relevant for KNXnet/IP devices:

Parameter:	NETIP.NetworkCardIPAddress
Scope:	IP address
Default value:	IP address of the default network interface
Unit:	None
Description:	It specifies the IP address of the network interface which is used to communicate with KNXnet/IP gateways. If no IP address is specified, the IP address of the system’s default network interface is used.

Parameter:	NETIP.NAT
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	This parameter specified whether the KNXnet/IP device is behind a NAT router or firewall. ON – the NAT flag within the KNXnet/IP tunneling protocol is set. OFF – the NAT flag is not set.

4.3.1.2. KNX group address definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaTelegramDefinitions.35.dat
```

Within this configuration files, the KNX datapoints i.e. the KNX group addresses are specified. The structure of the definition file is as follows:

```
'Syntax of the Telegram Definition Table:
'log.KNX Address; IP Address; Priority; Data Size; EIS Type; Signed/Unsigned; Unit; Description; Path; Control Data;
      Alias; Read on Reconnect; Read Cyclically Interval;
' Persistent;Historical;Synchronize;IN Converter;OUT Converter
1/0;BROADCAST;LOW;1BIT;EIS1;;;Telegram Demo EIS1;;;;F;0;F;F;F;;
2/0;BROADCAST;LOW;4BIT;EIS2;;;Telegram Demo EIS2;;;;F;0;F;F;F;;
3/0;BROADCAST;LOW;3BYTE;EIS3;;;Telegram Demo EIS4;;;;F;0;F;F;F;;
4/0;BROADCAST;LOW;3BYTE;EIS4;;;Telegram Demo EIS4;;;;F;0;F;F;F;;
5/1;BROADCAST;LOW;2BYTE;EIS5;;°C;Telegram Demo EIS5 1;;;;F;0;F;F;F;;
5/21;BROADCAST;LOW;2BYTE;EIS5;;mA;Telegram Demo EIS5 2;;;;F;0;F;F;F;;
```

Each line – except comment lines that start with ' – defines a single KNX group address.

> ! This definition file can also be generated using the NETxKNXConvertETS App which generates the KNX datapoint definitions from an exported ETS project. For more information see Section 3.4.1.

Column 1 – log.KNX Address
It defines the logical KNX group address of the datapoint. The KNX group address can be specified either in three level form (e.g. 0/2/1) or in tow level form (e.g. 0/513).

Column 2 – IP Address
It specifies the IP address of the KNXnet/IP device to which the specified KNX group address is assigned. Thus, it defines KNXnet/IP device that is used to send and receive KNX telegrams that are originated to this group address. The name “BROADCAST” defines a KNX datapoint which is sent to all defined gateways.

> ! The logical KNX group address together with the IP address of the KNXnet/IP device define the effective address of the KNX datapoint. This effective address is internally used by the server. Using this addressing scheme, it is possible to use the same KNX group address for multiple KNX datapoints as long as they are assigned to different KNXnet/IP devices. This means that behind each gateway an independent ETS address space can exist. Theoretically up to 1000 ETS projects with the same address space are possible within the system.

Column 3 – Priority
It defines the KNX priority that is used to send KNX group telegrams. Possible values are SYSTEM, HIGH, ALARM and LOW. The standard value “LOW” should be used in general.

> ! Columns 4 to 7 define how to interpret the received data of a KNX datapoint. At least two of these attributes has to defined to specify the semantic of a datapoint. All possible combinations are listed in the data type table (cf. Section 4.9.2).

Column 4 – Data Size
It defines the size of the datapoint. Possible values are: 1BIT, 2BIT, 4BIT, 1BYTE, 2BYTE, 3BYTE, 4BYTE, 10BYTE, 14BYTE.

Column 5 – EIS Type
Using this column, the data types is specify by defining the EIB Interworking Standard (EIS) type.

Column 6 – Signed/Unsigned
This attribute specifies whether the data type is a signed or unsigned one.

Column 7 – Unit
It defines engineering units of the KNX datapoint. It has to be selected from the list.

Column 8 – Description
This attribute can be used to specify a human-readable text that further describes the datapoint.

Column 9 – (deprecated) Path
This attribute is used for the automatic PDA visualization to create a tree structure like “Home/Level1/living room”.

Column 10 – (deprecated) Control Data
This attribute is used for additional input data for the automatic PDA visualization.

Column 11 – (deprecated) Alias
This attribute can be used to define an alias name for this KNX datapoint. This alias is shown within the server item tree under the sub tree “Alias” (cf. Section 1.3.2).

Column 12 – Read on Reconnect
This attribute is used to define whether a KNX Read telegram shall be sent when the corresponding KNXnet/IP gateway reconnects or when the server is started. If this functionality is activated, the read flag has to be set within the corresponding KNX device using ETS.

Column 13 – Read Cyclically Interval
This attributed is used to define whether cyclic KNX Read telegrams be sent to retrieve the current value of the datapoint. If this functionality is activated, the read flag has to be set within the corresponding KNX device using ETS. Note that cyclically reading KNX datapoints consumes bandwidth within the KNX lines. The allowed maximum Value is 65535 s with is identical to 18,20 h.

Column 14 – Persistent
This parameter specifies whether the value of the datapoint is persistent (restored from the database after server start up) or not.

Column 15 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not.

Column 16 – Synchronize
If this parameter is set to “True (T)”, the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

Column 17 – IN Converter
At this parameter you are able to define a LUA function, which can be used for INcoming conversion.

E.g. InConv()

Column 18 – OUT Converter
At this parameter you are able to define a LUA function, which can be used for OUTgoing conversion.

E.g. OutConv()

There is a own chapter with small overview of LUA functions implemented for the conversion (cf. Section 4.6.3).

4.3.1.3. KNX gateway definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaGatewayDefinitions.35.dat
```

The gateway manager maintains the connections to the KNXnet/IP devices which are connected to the KNX system. The gateway definition table defines all the KNXnet/IP interfaces and routers that are used within the project.

! No telegrams are sent to the offline gateways.

The structure of the gateway definition file is as follows:

```
'Syntax of the Gateway Definition Table:
'IP Address;Type;Port;Name;Locality;Description ;Options
'
192.168.1.1;IGS;51000; GATE 1;Room 51;pre-defined Gateway 1
192.168.1.2;NETIP;3671;Gate2;;;;;;;
192.168.1.3;EIBNODE;1634;Gate3;;;;;;;
```

Each line – except comment lines that start with ' – defines a single KNXnet/IP gateway.

! The virtual “BROADCAST” gateway is added by the system and must not be defined in the table.

Column 1 – IP Address
This attribute specifies the IP address of the KNXnet/IP gateway. The IP address is also used to uniquely identify

the gateway within the server. Therefore, it must be unique within the whole network.

Column 2 – Type
The type of the KNXnet/IP gateway. “IGS” refers to ABB IG/S, EIBNODE to b.a.b - technologie eibnode, and NETIP to all other KNXnet/IP routers and interfaces.

Column 3 – Port
This attributes represents the port number which is used to connect to the gateway.

Column 4 – Name
It is used to give the gateway a human-readable name (can be any user-defined string).

Column 5 – Locality
This attribute can be used to specify the location of the gateway (can be any user-defined string).

Column 6 – Description
This attribute can be used to specify a human-readable text that further describes the gateway (can be any user-defined string).

Column 7 – Options
If the gateway is of type “EIBNODE”, the SenderNetID has to be specified here.

Column 8 – 11 – Path, Extended Data 1 – 3
These columns are for internal use only.

4.3.1.4. KNX device definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaDeviceDefinitions.35.dat
```

The KNX device manager can check the availability of KNX devices. It tries to connect to the defined devices to verify whether they can be reached or not. If the connection fails, the value of the corresponding server item of the device is set to false. This feature is mainly used to check the availability for important devices. Note that the monitoring of KNX devices consumes network bandwidth within the KNX network, so it should only be used when necessary. Note that the telegrams that are used to verify the availability of KNX devices are not group telegrams and so they are not shown in the telegram monitor.

! The system parameter “SYS.EnableDeviceManager” has to be set to “ON”.

The device definition file specifies all the devices that shall be checked. The structure of this file is as follows:

```
'Syntax of the Device Definition Table:
'physical KNX Address;IP Address of Gateway;Polling interval (sec);Description;Alias
'
1.1.1;192.168.1.7;20;Device 1;Switch 1
1.1.2;192.168.1.8;40;Device 2;
```

Each line – except comment lines that start with ' – defines a single KNX device.

Column 1 – Physical KNX Address
This attribute specifies the physical KNX Address of the device which shall be monitored.

Column 2 – IP Address of Gateway
This attribute defines the IP address of the gateway via the device is reachable. “BROADCAST” is not supported in this case.

Column 3 – Polling interval
This attribute defines the polling interval in seconds.

Column 4 – Description
This attribute can be used to specify a human-readable text that further describes the device (can be any user-defined string).

Column 5 – Alias
It defines an alias name for the server item that represents the KNX device. This alias is shown within the server item tree under the sub tree “Alias” (cf. Section 1.3.2).

4.3.1.5. KNX group alias definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaGroupAliases.35.dat
```

Within this configuration files, the names for the KNX main and middle groups can be specified. The structure of the definition file is as follows:

```
'Syntax of the Group Alias Definition Table:
'KNX Main Group Number;KNX Middle Group Number;IP Address;Name
'
1;;192.168.1.2;Main Group 1
1;0;192.168.1.2;Middle Group 1/0
1;1;192.168.1.2;Middle Group 1/1
```

Each line – except comment lines that start with ' – defines a single KNX group address alias.

! This definition file can also be generated using the NETxKNXConvertETS App which generates the KNX group alias definitions from an exported ETS project. For more information see Section 3.4.1.

Column 1 – KNX Main Group Number
It defines the number of the KNX main group.

Column 2 – KNX Middle Group Number
It defines the number of the KNX middle group. If this field is left empty, the group alias entry is for the KNX main group address. Otherwise the alias is defined for a KNX middle group.

Column 3 – IP Address
It specifies the IP address of the KNXnet/IP device to which the specified KNX main or middle group address alias is defined.

Column 4 – Name
Here the name of alias is specified which shall be shown in the Server Item Tree. Note that this alias does not influence the Item ID of the KNX group address. The alias is set as description of the Server Branch that represents the main or middle group.

4.3.1.6. KNX event definitions

The KNX sub system of the NETx BMS Server provides a so called event processor. The event processor can be used to add KNX control functionality directly within the server without changing the functionality of the KNX installation (i.e. without modifying the ETS project). Using the KNX event processor, control functionality that is not available within the field and automation level of the already existing KNX installation can be added.

Within the current version of the NETx BMS Server, the following different kinds of events are supported for the KNX sub system:

1. Links: Links are used to associate one KNX group address with another one. This mechanism can be used to implement KNX datapoints that have more than one receiving KNX group address (cf. Section 4.3.1.6).
2. Response events: Response events make it possible to react on incoming KNX read or write requests. Based on user-defined conditions, KNX read or write request can be forwarded to other KNX communication group that are even assigned to a different KNX gateway. Using these mechanism, KNX group telegrams can be forwarded from one KNX communication group to another group that may be located in a different KNX network segment (cf. Section 4.3.1.6).
3. Timer events: A timer event provides the opportunity to send a KNX read request to a certain KNX datapoint or to send a predefined KNX datapoint value at a dedicated point in time (cf. Section 4.3.1.6).
4. Cyclic events: A cyclic event can be used to send a KNX read request to a certain KNX datapoint or to send a predefined KNX datapoint value in defined time intervals (cf. Section 4.3.1.6).

All these events are handled independent of the BMS Client. The BMS Clients are certainly informed about each change and accomplished action, so that they have the current data available in any case.

! Note that this event processor is only available for KNX datapoints. Furthermore, events are only triggered when there is a corresponding change of the KNX group address (i.e. a KNX group write or KNX group read) at the physical KNX medium – directly changing the server item within the server does not trigger an event. If general control functionality has to be added that is not limited to KNX and that also reacts on item changes that directly performed within the server, task definitions (cf. Section 4.5.2) or LUA scripting (cf. Section 4.6) has to be used.

Link configuration

Location:

```
<WorkspaceDirectory>\DataFiles\nxaLinkDefinitions.35.dat
```

The NETx BMS Server can be used to link one KNX group address with another KNX group address that is assigned to the same KNX gateway. Using this mechanism, it is possible implement KNX datapoints that have more than one receiving KNX group address.

! The link manager must be activated within the system configuration file of the server by setting the parameter “KNX.SetLinkOnReceive” to “ON” (cf. Section 4.2.2).

The concept “linked addresses” is explained at the following example:

Example:

Consider, for example, two KNX devices with the physical addresses 1.2.3 and 1.2.4 are defined within ETS.

The device 1.2.3 has a datapoint that has the sending KNX group address 3/0/2. In addition the two KNX group address 3/0/3 are assigned to this datapoint as receiving group address. The device 1.2.4 has a datapoint that has 3/0/3 as sending address.

This configuration within ETS has the following behaviour:

- If a group write is sent to 3/0/2, the value of the datapoint within the device 1.2.3 is changed.
- If a group write is sent to 3/0/3, the value of the datapoint within the device 1.2.4 as well as the datapoint within the device 1.2.3 is changed.

If no link definitions is specified within the NETx BMS Server, a group write to 3/0/3 will only update the Server Item with the KNX group address 3/0/3. However, within the KNX system the datapoint within the device 1.2.3 that has a sending group address of 3/0/2 is also updated since 3/0/3 is a “receiving group address” for this datapoint. As a result, there is an inconistancy between the virtual model within the NETx BMS Server and the real process image.

To solve this issue, the following KNX link definitions have to specified within the server:

```
3/0/3;BROADCAST;3/0/2
```

This link definition specifies that whenever a KNX value change is received with the destination group address 3/0/2, the Server Item that has the KNX group address 3/0/3 is also updated.

! Note that no KNX telegrams will be sent by link definitions since the data is only forwarded internally. Furthermore, it is important to note that links are only triggered by KNX telegrams. If the value of the link address is set internally by server (e.g. by a BMS Client), the value the associated link addresses will not be changed since the real value was not changed within the KNX system

The structure of the link definition file is as follows:

```
'Syntax of the Link Definition Table:
'<log.KNXAddress>;<IP Address>;<linked log.KNXAddress 1>;<linked log.KNXAddress 2>;... <linked log.KNXAddress n>
'
'Generated by NETxKNX ETS Converter 4.0.22
'Generated at 20.03.2012 13:49:08
'
03/0/003;192.168.1.13;03/0/002;03/0/001
05/0/003;192.168.1.13;03/0/002
```

Each line – except comment lines that start with ' – defines a single link.

! This definition file can also be generated using the NETxKNXConvertETS App which generates the KNX link definitions from an exported ETS project. For more information see Section 3.4.1.

Column 1 – Master Link

This attribute specifies the KNX group address that causes a change of the group addresses that are dedicated to this link. This means that if a KNX group read or write is received with a destination address that is equal to

this address, the server updates all linked group addresses. Note that multiple links that use the same master link address can be defined.

Column 2 – IP Address
It specifies the IP address of the gateway to which the link is limited. If the “BROADCAST” gateway is used here, the master link address and all sub link addresses have to be defined as “BROADCAST”, too.

Column 3 – 1000 – Sub Links
The remainder columns specify the group address that have to be updated if a KNX group read or write occurs at the specified master link address.

Response Event configuration
Location:

```
<WorkspaceDirectory>\EventFiles\nxaResponeEvents.35.dat
```

Response events are used to react to incoming KNX group read or write requests by sending KNX read or writes to another KNX group address. Compared to link definitions, the source and destination group addresses of response events can be assigned to different KNX gateways.

In addition, for each defined response event, conditions can be specified that have to be fulfilled to handle the event. Conditions can be based on the value of the incoming event (i.e. equal, lower, or greater than a pre-defined value) or on the point in time when the event occurs.

The structure of the response event file is as follows:

```
' Active;Name;AddressToSend;IPAddressToSend;TelegramType;SendReceivedValue;ValueToSend;AddressToMonitor;
      IPAddressToMonitor;
' EqaulValueToMonitor;LessValueToMonitor;GreaterValueToMonitor;UseWorkingTime;WorkingTimeBegin;WorkingTimeEnd;
' UseCalendar;ActiveWeekDays;BeginMonthDay;BeginMonth;EndMonthDay;EndMonth;Path;Extended Data...
'
T;Test1;1/2;BROADCAST;WRITE;F;1;6/1;BROADCAST;;;13;T;8:00;17:00;T;FTTTTTF;5;15;2;8;;;;
F;Test2;1/3;BROADCAST;WRITE;F;5;6/1;BROADCAST;;12;F;;;F;;;;;;;;;;
```

Each line – except comment lines that start with ' – defines a single response event.

Column 1 – Active
It defines whether this event is enabled. If the value is “False (F)”, this event will not be considered by the server.

Column 2 – Event Name
This attribute specifies the name of the event that has to be unique within the server. The event name is used for displaying purposes. Up to 64 characters are allowed for event names.

Column 3 – KNX Group Address to Send
This attribute defines the KNX group address that is used to send the reaction on the specified event.

Column 4 – IP Address to Send
Here, the IP address of the KNX gateway that is used to send the reaction is specified.

Column 5 – Telegram Type to Send
It defines the type of telegram to send as a reaction on the event. Possilbe value are “WRITE” for sending a KNX group write and “READ” for sending a KNX group read. If the type is set to “READ”, the values of the columns “Send Received Value” and “Value to Send” are ignored.

Column 6 – Send Received Value
This column defines whether a predefined or the received value shall be sent to the destination group address. If it is set to “T”, the KNX group telegram is sent with the value of the telegram that caused the event. In this case the next column will be ignored. If it is set to “F”, the value from the column “Value to Send” will be sent.

Column 7 – Value to Send
It defines the value of the telegram. If the value in the column “Send Received Value” is “F”, this value will be sent as KNX-telegram to the destination address.

Column 8 – KNX Group Address to Monitor
This column defines the KNX group address that triggers the event. If, for example, the address 1/1/1 is entered and a KNX telegram with this address is received, this event will be triggered. Because a KNX datapoint address consists of the KNX group address and the IP address of the KNX gateway to which the group address is assigned, the “IP-address” is also considered.

Column 9 – IP Address to Monitor
It defines the IP address part of the KNX datapoint address which is to be monitored.

Column 10 – Equal Value to Monitor
The event is only triggered if the received value is equal to the specified one. If this column is left empty, no comparison will be made.

Column 11 – Less Value to Monitor
The event is only triggered if the received value is less that the specified one. If this column is left empty, no comparison will be made.

Column 12 – Greater Value to Monitor
The event is only triggered if the received value is greater that the specified one. If this column is left empty, no comparison will be made.

Column 13 – Use Working Time
This column defines if the working time is to be considered. The value “T” defines that the succeeding columns are used to define the working time when the event is active. The event will be executed, if the actual time is between “Working Time Begin” and “Working Time End”. If this column is set to “F” or it is left empty, the event monitoring will be active the whole time.

Example: “. . . ;T;8:00;17:00;. . . ” specifies that the event is only active within 8:00 and 17:00.

Column 14 – Working Time Begin
It defines the beginning of the working time.

Column 15 – Working Time End
It defines the end of the working time.

Column 16 – Use Calendar
This column defines if the specified event shall be limited to a specific time period. This time interval can be specified based on a calendar-like filter. The value “T” defines that the succeeding columns are used to define the time period when the event is active. If this column is set to “F” or it is left empty, the calendar functionality will not be used.

Example: “. . . ;T;FTTTTTF;10;1;15;1;. . . ” specifies that the event is only active from Monday to Friday from 10th January to 15th of January

Column 17 – Active Weekdays
This column defines that this event shall only be active on specific week days. The entry consists of exactly seven “F” (event not active) or “T” (event active) characters. The first character specifies the behaviour for Sunday, the second one for Monday, and so on.

Example: “FTTTTTF” – this filter defines that this event is not active on Sundays and Saturdays.

Column 18 – Month Day Begin
This column determines from which day of the month this event is active. It applies only together with the value of the Column “Month Begin”.

Column 19 – Month Begin
This column determines from which month this event is active. It applies only together with the value of the Column “Month Day Begin”.

Column 20 – Month Day End
This column defines up to which day of the month this event is active. It applies only together with the value of the Column “Month End”.

Column 21 – Month End
This column defines up to which month this event is active. It applies only together with the value of the Column “Month Day End”.

Column 22 – 25 – Path, Extended Data 1 – 3
These columns are for internal use only.

Timer Event configuration
Location:

```
<WorkspaceDirectory>\EventFiles\nxaTimerEvents.35.dat
```

Timer events are used to send KNX group read or write requests at dedicated points in time. In addition, for each defined timer event, conditions based on the point in time when the event occurs can be specified.

The structure of the timer event file is as follows:

```
' Active;Name;AddressToSend;IPAddressToSend;TelegramType;ValueToSend;TimePoint;
' UseCalendar;ActiveWeekDays;BeginMonthDay;BeginMonth;EndMonthDay;EndMonth;Path;Extended Data...
'
'
T;Test1;14/1;192.168.1.1;WRITE;1;17:18:30;;;;;;;;;;
F;Test2;14/1;BROADCAST;WRITE;0;16:15:35;;;;;;;;;;
```

Each line – except comment lines that start with ' – defines a single timer event.

Column 1 – Active
It defines whether this event is enabled. If the value is "F", this event will not be considered by the server.

Column 2 – Event Name
This attribute specifies the name of the event that has to be unique within the server. The event name is used for displaying purposes. Up to 64 characters are allowed for event names.

Column 3 – KNX Group Address to Send
This attribute defines the KNX group address that is used to send the reaction on the specified event.

Column 4 – IP Address to Send
Here, the IP address of the KNX gateway that is used to send the reaction is specified.

Column 5 – Telegram Type to Send
It defines the type of telegram to send as a reaction on the event. Possible value are "WRITE" for sending a KNX group write and "READ" for sending a KNX group read. If the type is set to "READ", the value of the column "Value to Send" is ignored.

Column 6 – Value to Send
It defines the value of the telegram.

Column 7 – Time Point to Send at
This attribute specifies the point in time when the reaction telegram has to be sent. The encoding of the point in time is hh:mm or hh:mm:ss.

Column 8 – Use Calendar
This column defines if the specified event shall be limited to a specific time period. This time interval can be specified based on a calendar-like filter. The value "T" defines that the succeeding columns are used to define the time period when the event is active. If this column is set to "F" or it is left empty, the calendar functionality will not be used.

Example: "...;T;FTTTTTF;10;1;15;1;..." specifies that the event is only active from Monday to Friday from 10th January to 15th of January

Column 9 – Active Weekdays
This column defines that this event shall only be active on specific week days. The entry consists of exactly seven "F" (event not active) or "T" (event active) characters. The first character specifies the behaviour for Sunday, the second one for Monday, and so on.

Example: "FTTTTTF" – this filter defines that this event is not active on Sundays and Saturdays.

Column 10 – Month Day Begin
This column determines from which day of the month this event is active. It applies only together with the value of the Column "Month Begin".

Column 11 – Month Begin
This column determines from which month this event is active. It applies only together with the value of the Column "Month Day Begin".

Column 12 – Month Day End
This column defines up to which day of the month this event is active. It applies only together with the value of the Column "Month End".

Column 13 – Month End
This column defines up to which month this event is active. It applies only together with the value of the Column "Month Day End".

Column 14 – 17 – Path, Extended Data 1 – 3

These columns are for internal use only.

Cyclic Event configuration
Location:

```
<WorkspaceDirectory>\EventFiles\nxaCyclicEvents.35.dat
```

Cyclic events are used to send KNX group read or write requests at dedicated time intervals. In addition, for each defined cyclic event, conditions based on the point in time when the event occurs can be specified.

The structure of the cyclic event file is as follows:

```
' Active;Name;AddressToSend;IPAddressToSend;TelegramType;ValueToSend;TimePoint;
' UseCalendar;ActiveWeekDays;BeginMonthDay;BeginMonth;EndMonthDay;EndMonth;Path;Extended Data...
'
'
T;Test1;14/1;192.168.1.1;WRITE;1;17:18:30;;;;;;;;;;
F;Test2;14/1;BROADCAST;WRITE;0;16:15:35;;;;;;;;;;
```

Each line – except comment lines that start with ' – defines a single cyclic event.

Column 1 – Active
It defines whether this event is enabled. If the value is “F”, this event will not be considered by the server.

Column 2 – Event Name
This attribute specifies the name of the event that has to be unique within the server. The event name is used for displaying purposes. Up to 64 characters are allowed for event names.

Column 3 – KNX Group Address to Send
This attribute defines the KNX group address that is used to send the reaction on the specified event.

Column 4 – IP Address to Send
Here, the IP address of the KNX gateway that is used to send the reaction is specified.

Column 5 – Telegram Type to Send
It defines the type of telegram to send as a reaction on the event. Possible value are “WRITE” for sending a KNX group write and “READ” for sending a KNX group read. If the type is set to “READ”, the value of the column “Value to Send” is ignored.

Column 6 – Value to Send
It defines the value of the telegram to be send.

Column 7 – Time Interval
This attribute specifies the time interval in seconds for sending the KNX telegrams of the defined event.

Column 8 – Use Calendar
This column defines if the specified event shall be limited to a specific time period. This time interval can be specified based on a calendar-like filter. The value “T” defines that the succeeding columns are used to define the time period when the event is active. If this column is set to “F” or it is left empty, the calendar functionality will not be used.

Example: “. . . ;T;FTTTTTF;10;1;15;1;. . . ” specifies that the event is only active from Monday to Friday from 10th January to 15th of January

Column 9 – Active Weekdays
This column defines that this event shall only be active on specific week days. The entry consists of exactly seven “F” (event not active) or “T” (event active) characters. The first character specifies the behaviour for Sunday, the second one for Monday, and so on.

Example: “FTTTTTF” – this filter defines that this event is not active on Sundays and Saturdays.

Column 10 – Month Day Begin
This column determines from which day of the month this event is active. It applies only together with the value of the Column “Month Begin”.

Column 11 – Month Begin
This column determines from which month this event is active. It applies only together with the value of the Column “Month Day Begin”.

Column 12 – Month Day End
This column defines up to which day of the month this event is active. It applies only together with the value of the

Column “Month End”.

Column 13 – Month End
This column defines up to which month this event is active. It applies only together with the value of the Column “Month Day End”.

Column 14 – 17 – Path, Extended Data 1 – 3
These columns are for internal use only.

4.3.1.7. ETS object definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaKNXObjectDefinitions.dat
```

Within this configuration file, the KNX communication objects as represented in the item tree are specified. The structure of the definition file is as follows:

```
'Syntax of the ETS Object Definition Table:
'Name;Object function;Device name;Path;Communication flag;Read flag;Write flag;Transmit flag;Update flag;Read on init
      flag;Physical address;IP address;Group addresses (first one is sending)
Output A;10: Switch;Room1 - Light HVAC Actuator;Demo_Case_Large\Buildings\BuildingA\Floor1\Cabinet1\3.1.1;T;F;T;F;F;F
      ;3.1.1;192.168.1.15;0/0/1@192.168.1.15
Output A;29: Status Switch;Room1 - Light HVAC Actuator;Demo_Case_Large\Buildings\BuildingA\Floor1\Cabinet1\3.1.1;T;T;F
      ;T;F;F;3.1.1;192.168.1.15;0/0/2@192.168.1.15
Output B;30: Switch;Room1 - Light HVAC Actuator;Demo_Case_Large\Buildings\BuildingA\Floor1\Cabinet1\3.1.1;T;F;T;F;F;F
      ;3.1.1;192.168.1.15;0/0/3@192.168.1.15
Output B;49: Status Switch;Room1 - Light HVAC Actuator;Demo_Case_Large\Buildings\BuildingA\Floor1\Cabinet1\3.1.1;T;T;F
      ;T;F;F;3.1.1;192.168.1.15;0/0/4@192.168.1.15
```

Each line – except comment lines that start with ' – defines a single communication object.

> ! The ETS object definitions within this file are automatically created by the NETx BMS App. Therefore, editing it manually is not necessary in general.

Column 1 – Name
This parameter refers to the name of the communication object.

Column 2 – Object function
Here the name of the function of the communication object is specified.

Column 3 – Device name
This is the device name to which the communication object belongs to.

Column 4 – Path
This column defines the path within the ETS view where the communication object is shown.

Column 5 – Communication flag
The KNX communication flag of the communication object. For more information see ETS documentation.

Column 6 – Read flag
The KNX read flag of the communication object. For more information see ETS documentation.

Column 7 – Write flag
The KNX write flag of the communication object. For more information see ETS documentation.

Column 8 – Transmit flag
The KNX transmit flag of the communication object. For more information see ETS documentation.

Column 9 – Update flag
The KNX update flag of the communication object. For more information see ETS documentation.

Column 10 – Read on init flag
The KNX read on init flag of the communication object. For more information see ETS documentation.

Column 11 – Physical address
The physical address of the device where the communication object is located.

Column 12 – Reserved
Not used.

Column 13 – Group addresses (first one is sending)
The KNX group addresses that are linked to the communication object. The first group address is the sending group address i.e. the address which is used for send whenever the communication object changes its value.

4.3.2. Modbus XIO Interface

The Modbus interface is used to integrate Modbus datapoints into the NETx BMS Server via Modbus TCP. The configuration files of the Modbus module are described in the remainder of this section.

! Compared to BACnet and KNX, there is no unique way to get the configuration data of the Modbus devices and datapoints. To get the necessary information about the Modbus configuration, please refer to the data sheets and manuals of the used Modbus devices.

! Some vendors of Modbus devices provide their own engineering tool. Most of these tools have an export functionality. The output of this export function (e.g. a text file) can be converted to configuration format of the NETx BMS Server.

4.3.2.1. Modbus device definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.Modbus.DeviceDefinitions.dat
```

The Modbus device manager checks the availability of Modbus TCP devices. It tries to connect to the defined devices and verifies whether they can be reached or not. If a connection to a Modbus device fails, the value of the corresponding Server Item that represents the device within the server is set to false. After having successfully established a connection to the Modbus device, this connection is continuously monitored by using a heart-beating mechanism. If a Modbus device is not reachable anymore, the corresponding Server Item is set to false and all Modbus datapoint that are assigned to this device are set to invalid.

The structure of the Modbus device definition file is as follows:

```
' Modbus device configuration file
'DeviceName;IPAddress;Port;UnitIdentifier;Description;Endianess;Wordswap;DWordSwap;BitSwap;Max_parallel;Tasktimeout
ModbusController1;192.168.1.20;502;250;59122_104_401;B;T;T;F;1;5000
```

Each line – except comment lines that start with ' – defines a single Modbus device.

Column 1 – DeviceName
The name of the device. For all Modbus devices, this name has to be unique since it is used as unique identifier for the Modbus device.

Column 2 – IP Address of Device
This defines the IP address of the Modbus device via which the device is reachable.

Column 3 – Port
This defines the port number which is used to open the Modbus TCP connection.

Column 4 – UnitIdentifier
This identifier addressed the Modbus unit within the controller. This identifier is used to communicate via devices such as Modbus bridges, routers, and gateways that use a single IP address to support multiple independent Modbus end units. For setting this parameter, please refer to the data sheet of the Modbus device.

Column 5 – Description
This attribute can be used to specify a human-readable text that further describes the device.

Column 6 – Endianess
This vendor-specific parameter defines the endianess of the memory blocks that are used within the device. If this parameter is set to “S”, little(small) endian is used within the Modbus device. Otherwise, big endian is used which is also the default in Modbus. For setting this parameter, please refer to the data sheet of the Modbus device.

Column 7 – Wordswap
This vendor-specific parameter defines whether Words are swapped (set to “T” – default in Modbus) or not (set to “F”). For setting this parameter, please refer to the data sheet of the Modbus device.

Column 8 – DWordswap
This vendor-specific parameter defines whether Double Words are swapped (set to “T” – default in Modbus) or not (set to “F”). For setting this parameter, please refer to the data sheet of the Modbus device.

Column 9 – Bitswap
This vendor-specific parameter defines whether the bits are swapped (set to “T”) or not (set to “F” – default in Modbus). For setting this parameter, please refer to the data sheet of the Modbus device.

Column 10 – Max_parallel
This attribute specifies the maximum amount of requests that are sent to the device in parallel. The default value is “1”.

Column 11 – Request Timeout
This attribute specifies the timeout interval in milliseconds that is used to determine whether a Modbus request failed or not. The default is 5000 ms.

Column 12 – Retry Count
This parameter specifies the amount of retries before a Modbus request is declared as failed.

Column 13 – Retry Delay
The Retry Delay specifies the timeout between a failed Modbus request and resending it again.

Column 14 – Write Mode
This parameter specifies how the NETx BMS Server writes Modbus data. Valid values are “Multi” (default), “Single”, and “SingleReverse”. “Multi” means that the Modbus driver tries to combine write requests with subsequent addresses into one request (when it is possibel). “Multi” uses the Modbus function codes 15 and 16. “Single” means that each Modbus coil or register is written separately using function code 5 and 6. “SingleReverse” also uses single requests but writes datapoints that contains multiple coils or registers in reverse order.

Column 15 – Address Mapping
To use an addressing scheme different to the standard memory addresses starting with address 0, an address mapping is possible. The address mapping itself is specified in the Address Mapping Definition file (cf. Section 4.3.2.3). Use an address mapping, the name of the mapping that is used in the Address Mapping Definition file has to be specified here.

Column 16 – Ignore Quality For Datapoints
If set to true, the quality of datapoints are not set to “BAD” if the connection to the devices fails or timed-out. The default value is false.

Column 17 – Use Device Identification Request
If set to true, the Modbus interface sends “Read Device Identification” requests (function code 43) in order to check the availability of the Modbus device. The default value is false.

Column 18 – Refresh Interval
If “Use Device Identification Request” is set to true, this parameter specifies the interval between two “Read Device Identification” requests.

4.3.2.2. Modbus datapoint definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.Modbus.DatapointDefinitions.dat
```

The Modbus interface provides the opportunity to read and write Modbus datapoints. The current version of the NETx BMS Server 2.0 support the handling of Modbus discrete inputs, coils, holding registers, and input registers. For more information see Section 4.9.3.

The encoding of Modbus datapoint types is vendor-specific. To be able to deal with the different vendor-specific encoding, configuration parameters to set the swapping of Words, Double Words, and Bits are available. These parameters can be defined for each device and for each datapoint. Furthermore, it is possible to specify the endianess that is used within the device (cf. Section 4.3.2.1).

Note that using the correct encoding settings is at utmost importance. If these parameters are incorrect, the corresponding values of the Modbus datapoints are interpreted, stored, and displayed in a wrong way. Furthermore, it has to be considered that these parameters are affecting each other. To choose the correct parameters please refer to the manual and datasheet of the used Modbus devices.

The structure of the Modbus datapoint definition file is as follows:

```
' Modbus datapoint configuration file
' DeviceName;ModbusType;Address;SubDataType;Size;Description;PollingInterval;Persistent;Historical;Synchronize;
      Wordswap;DWordswap;Bitswap
ModbusController1;discrete input;12416;bool;1;DI 1;10;F;F;F;F;T;F
ModbusController1;coil;12417;bool;1;DO 1;10;F;F;F;;;;;
ModbusController1;input register;12418;uint16;1;AI 1;10;F;F;F;F;;;;;
ModbusController1;holding register;12419;uint16;1;AO 1;10;F;F;F;F;;;;;
```

Each line – except comment lines that start with ' – defines a single Modbus datapoint.

Column 1 – DeviceName
This attribute refers to the name of the device which holds the datapoint. This name must match the device name used in the Modbus device configuration.

Column 2 – ModbusType
This parameter refers to the datapoint type. Valid types are defined in Section 4.9.3.

Column 3 – Address
This attribute defines the memory address of the Modbus datapoint.

Column 4 – SubDataType
This identifier specifies the data type of Server Item that shall represent the Modbus datapoint. Valid sub types are defined in Section 4.9.3.

Column 5 – Size
This identifier specifies the size of the Modbus datapoint (only considered if DataType is set to “string” or “wstring”, see Section 4.9.3.

Column 6 – Description
This attribute can be used to specify a human-readable text that further describes the datapoint.

Column 7 – PollingInterval
This parameter specifies the interval that is used to poll the value of the datapoint.

Column 8 – Persistent
This parameter specifies whether the value of the datapoint is persistent (restored from the database after server start up) or not.

Column 9 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not.

Column 10 – Synchronize
If this parameter is set to “T”, the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

Column 11 – Wordswap
This parameter can be used to overwrite the Word Swap parameter of the Modbus device. If this parameter is set to “T” (“TRUE”) or “F” (“FALSE”), the word swapping for this datapoint is overwritten. If this parameter is empty (or set to a value unequal to “T” or “F”), the word swapping specified in the device definition is used.

Column 12 – DWordswap
This parameter can be used to overwrite the Double Word Swap parameter of the Modbus device. If this parameter is set to “T” (“TRUE”) or “F” (“FALSE”), the double word swapping for this datapoint is overwritten. If this parameter is empty (or set to a value unequal to “T” or “F”), the double word swapping specified in the device definition is used.

Column 13 – Bitswap
This parameter can be used to overwrite the Bit Swap parameter of the Modbus device. If this parameter is set to “T” (“TRUE”) or “F” (“FALSE”), the bit swapping for this datapoint is overwritten. If this parameter is empty (or set to a value unequal to “T” or “F”), the bit swap specified in the device definition is used.

Column 14 – Read on Reconnect
If this parameter is set to “T”, a read request is sent The default value is “T”.

Column 15 – Write Mode
If not specified, the write mode that is defined within the Modbus Device Definition is used.

4.3.2.3. Modbus address mapping definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.Modbus.AddressMappingDefinitions.dat
```

To use an addressing scheme different to the standard Modbus memory addresses that start with address 0, an address mapping is possible. These additional address mappings can be specified here. For each Modbus device, an individual address mapping can be specified (cf. Section 4.3.2.1).

The structure of the Modbus address mapping definition file is as follows:

```
' Modbus address mapping definition file
'
' Mapping Name;Offset for Discrete Input;Offset for Coil;Offset for Input Register;Offset for Holding Register
RegisterNumber;1;1;1;1
FunctionCode;20001;10001;40001;30001
```

Each line – except comment lines that start with ' – defines one Modbus address mapping.

Column 1 – Mapping Name
Here the name of the mapping has to be specified. This name is used within the Modbus Device Definition file to identify the mapping. Therefore, the mapping name has to be unique.

Column 2 – Offset for Discrete Input
This offset specifies the number that is subtracted from the Discrete Input addresses that are used in the Modbus Datapoint Defintion file. E.g. if the offset is 20001 and the datapoints address 20010 is used, the Modbus interfaces reads the Discrete Input at memory address 9.

Column 3 – Offset for Coil
This offset specifies the number that is subtracted from the Coil addresses that are used in the Modbus Datapoint Defintion file. E.g. if the offset is 20001 and the datapoints address 20010 is used, the Modbus interfaces reads the Discrete Input at memory address 9.

Column 4 – Offset for Input Register
This offset specifies the number that is subtracted from the Input Register addresses that are used in the Modbus Datapoint Defintion file. E.g. if the offset is 20001 and the datapoints address 20010 is used, the Modbus interfaces reads the Input Register at memory address 9.

Column 5 – Offset for Holding Register
This offset specifies the number that is subtracted from the Holding Register addresses that are used in the Modbus Datapoint Defintion file. E.g. if the offset is 20001 and the datapoints address 20010 is used, the Modbus interfaces reads the Holding Register at memory address 9.

4.3.3. BACnet XIO Interface

The BACnet interface is used to integrate BACnet data (e.g. BACnet object, properties, and devices) into the NETx BMS Server via the BACnet/IP protocol. In addition, the BACnet interface can also be used to map Server Items to BACnet Objects. These created BACnet Objects can be accessed by any foreign BACnet client. Therefore, the NETx BMS Server provides both the BACnet Client and Server interface. The configuration files of the BACnet interface are described in the remainder of this section.

! Instead of editing the configuration files manually, the configuration of the BACnet devices and objects can be done by using the tool “BACnetExplorer”. For more information about this tool see Section 3.4.2).

4.3.3.1. BACnet configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\xio.BACnet.cfg
```

Within this configuration files, general configuration parameters of the BACnet interface are set. Each parameter within the BACnet configuration file is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

Parameter:	**XIO.BACnet.IPAddress**
Scope:	IP address
Default value:	127.0.0.1
Unit:	None
Description:	Specifies the IP address of the local network interface which is used to communicate with the BACnet devices.

! The IP address is also used by the BACnetExplorer. If the IP address is not valid or if it is set to 127.0.0.1, the driver tries to determine the IP address of your system. If more than one network interfaces are found, the first one is chosen. In addition, you can choose the IP address via the BACnetExplorer which then automatically stores the value within this configuration file.

Parameter:	**XIO.BACnet.Port**
Scope:	0 – 65535
Default value:	47808
Unit:	None
Description:	Specifies the port of the local network interface which is used to communicate with the BACnet devices and clients. If 0 is used or if the parameter is commented out, the server chooses a dynamic port.

Parameter:	**XIO.BACnet.RefreshInterval**
Scope:	0, 100 – 10000
Default value:	1000
Unit:	None
Description:	This parameter specifies the interval in milliseconds which is used to poll the BACnet devices. "'0"' deactivates device polling.

Parameter:	**XIO.BACnet.DefaultWritePriority**
Scope:	1 – 16
Default value:	16
Unit:	None
Description:	This parameter defines the BACnet Write Priority which is used by the server to write BACnet properties.

Parameter:	**XIO.BACnet.DeviceObjectIdentifier**
Scope:	0 – 4194303
Default value:	4194303
Unit:	None
Description:	Here, the BACnet device ID of the server can be specified. This ID must be unique within the BACnet network and must be different from the device IDs of all other BACnet devices.

Parameter:	**XIO.BACnet.DeviceObjectName**
Scope:	string
Default value:	None
Unit:	NETx BMS Server
Description:	Here, the BACnet device name of the server can be specified. This name must be unique within the BACnet network and must be different from the device names of all other BACnet devices.

Parameter:	**XIO.BACnet.Description**
Scope:	string
Default value:	None
Unit:	None
Description:	This parameter specifies a human-readable description of the server that is visible via the server's BACnet device object.

Parameter:	**XIO.BACnet.Location**
Scope:	string
Default value:	None
Unit:	None
Description:	This parameter refers to the "Location" property of the server's BACnet device object.

Parameter:	**XIO.BACnet.Logging**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Turns the logging for log level "Info" ON or OFF (for testing purposes only).

Parameter:	**XIO.BACnet.ModelName**
Scope:	string
Default value:	None
Unit:	None
Description:	This parameter refers to the “ModelName” property of the server’s BACnet device object.
Parameter:	**XIO.BACnet.APDUTimeout**
Scope:	1 – 30000
Default value:	1000
Unit:	milliseconds
Description:	This parameter specifies the BACnet APDU Timeout in milliseconds for resending message.
Parameter:	**XIO.BACnet.APDURetries**
Scope:	1 – 5
Default value:	3
Unit:	None
Description:	Here, the maximum ADPU transmission retries can be configured before the device is set to unavailable.
Parameter:	**XIO.BACnet.ResubscribtionInterval**
Scope:	1 – 2147483647
Default value:	3600
Unit:	None
Description:	This parameter specifies the interval in seconds at which the BACnet subscriptions are renewed. Default is 3600 s i.e. 1 hour.
Parameter:	**XIO.BACnet.InAlarmIsGood**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Set ON if BACnet objects that are in state “InAlarm” shall be displayed as GOOD.
Parameter:	**XIO.BACnet.FaultIsGood**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Set ON if BACnet objects that are in state “Fault” shall be displayed as GOOD.
Parameter:	**XIO.BACnet.OverriddenIsGood**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Set ON if BACnet objects that are in state “Overridden” shall be displayed as GOOD.
Parameter:	**XIO.BACnet.OutOfServiceIsGood**
Scope:	ON, OFF
Default value:	OFF
Unit:	None
Description:	Set ON if BACnet objects that are in state “OutOfService” shall be displayed as GOOD.
Parameter:	**XIO.BACnet.UseAddendum2008w**
Scope:	TRUE, FALSE
Default value:	TRUE
Unit:	None
Description:	Set TRUE if the new BACnet object types defined in BACnet Addendum 2008-w shall be used. If set to FALSE, Server items that shall be represented to BACnet objects are mapped to the standard object types (Analog Output, Analog Input, ...). Note setting this option to FALSE may lead to a loss of precision if the data size of the chosen server items is larger than the data size in the BACnet object.

Parameter:	XIO.BACnet.DCCPassword
Scope:	string
Default value:	None
Unit:	None
Description:	This parameter is used to set the password for allowing the use of the BACnet Device Control Communication services. In order to use these services, BACnet Clients must set this password within the corresponding requests.

Parameter:	XIO.BACnet.UnsolicitedCOV
Scope:	TRUE, FALSE
Default value:	FALSE
Unit:	None
Description:	If activated, COV notifications are processed even if no valid COV subscription is available within the NETx BMS Server. This option provides the opportunity to update BACnet object values for unsolicited COV notifications.

Parameter:	XIO.BACnet.UseBBMD
Scope:	TRUE, FALSE
Default value:	FALSE
Unit:	None
Description:	Set true if the NETx BMS Server shall register to a BBMD as foreign device.

Parameter:	XIO.BACnet.BBMDIPAddress
Scope:	IP address
Default value:	None
Unit:	None
Description:	If “UseBBMD” is set to true, the IP address of the remote BBMD has to be specified here.

Parameter:	XIO.BACnet.BBMDPort
Scope:	0 – 65535
Default value:	47808
Unit:	None
Description:	If “UseBBMD” is set to true, the port of the remote BBMD has to be specified here.

Parameter:	XIO.BACnet.BBMDTimeout
Scope:	1 – 32767
Default value:	3600
Unit:	None
Description:	If “UseBBMD” is set to true, the lifetime (in seconds) of the BBMD registration can be specified here.

4.3.3.2. BACnet device definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.BACnet.DeviceDefinitions.dat
```

The BACnet device manager checks the availability of BACnet/IP devices. It tries to connect to the defined devices and verifies whether they can be reached or not. If a connection to a BACnet device fails, the value of the corresponding Server Item that represents the device within the server is set to false. After having successfully established a connection to the BACnet device, this connection is continuously monitored by using a heart-beating mechanism. If a BACnet device is not reachable anymore, the corresponding Server Item is set to false and all BACnet data points that are assigned to this device are set to UNCERTAIN.

The structure of the BACnet device definition file is as follows:

```
' BACnet device configuration file
' Device ID;Device name;Net ID;Link layer address;Remote address;APDU size;Max parallel cmds
257653;BACnet Controller 1;0;192.168.0.22.186.192;;1476;1
2098178;BACnet Controller 2;5;192.168.0.23.186.192;240;480;1
```

Each line – except comment lines that start with ' – defines a single BACnet device.

! The BACnet device definitions within this file can automatically be set by using the BACnetExplorer. Therefore, editing it manually is not necessary in general.

Column 1 – Device ID
This parameter specifies the BACnet Device ID of the BACnet device. It refers to the BACnet Object ID of the BACnet Device.

Column 2 – Device name
This parameter specifies the BACnet Device name of the BACnet device. It refers to the BACnet Object Name of the BACnet Device.

Column 3 – Net ID
This attribute refers to the BACnet Network ID where the device is located. If "0" is used, the BACnet device is located within the same network as the NETx BMS Server.

Column 4 – Link layer address
This identifier specifies the BACnet data link address of the BACnet device. The exact format of this address information depends on the used BACnet network medium. The general syntax is: <address-part1>.<address-part2>.<address-part2>. . . For IP networks, the data link address is the IP address concatenated with the UDP port number. For example, if the BACnet device has the IP 192.168.1.4 and the default BACnet UDP port (47808) is used, the address has to be encoded as: 192.168.1.4.186.192 (47808 = hex BA C0 = 186.192).

Column 5 – Remote address
If the BACnet device is located in a remote BACnet network (i.e. not within the same network segment), the remote address specifies the address of the next hop router. The encoding is the same as for the BACnet data link address (Link layer address).

Column 6 – APDU size
This attribute is the maximum APDU size that is excepted by the remote BACnet device.

Column 7 – Max parallel cmds
This attribute specifies the maximum amount of requests that are sent to the device in parallel. The default value is "1".

4.3.3.3. BACnet object definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.BACnet.ObjectDefinitions.dat
```

The BACnet interface provides the opportunity to read and write BACnet Objects and their BACnet Properties. For a complete overview of supported BACnet Objects see Section 4.9.4.

The structure of the BACnet object definition file is as follows:

```
' BACnet object configuration file
' DeviceId;ObjectId;ObjectType;ObjectName;WritePriority;PollingInterval;Persistent;Historical;Synchronize;COVType
257653;0;Binary Input;BINARY_INPUT_0;15;5000;F;F;F
257653;0;Binary Output;BINARY_OUTPUT_0;15;5000;F;T;F
2098178;1;Analog Input;B'AI1;15;5000;F;T;F
2098178;1;Analog Output;B'AO1;15;5000;F;F;F
2098178;1;Analog Value;B'AVAL1;15;5000;F;F;F
2098178;1;Binary Value;B'BVAL1;15;5000;F;F;F
2098178;2;Accumulator;B'ACC1;15;5000;F;F;F
2098178;2;Pulse Converter;B'PC1;15;5000;F;F;F
2098178;1;Multi-State Input;B'MSI1;15;5000;F;F;F
2098178;1;Multi-State Output;B'16;15;5000;F;F;F
2098178;1;Multi-State Value;B'17;15;5000;F;F;F
```

Each line – except comment lines that start with ' – defines a single BACnet object.

! The BACnet object definitions within this file can automatically be set by using the BACnetExplorer. Therefore, editing it manually is not necessary in general.

Column 1 – DeviceId
The BACnet Device ID of the device which holds the BACnet Object. This name must match the device ID used in the BACnet device configuration.

Column 2 – ObjectId
This parameter refers to the BACnet Object ID of the defined object.

Column 3 – ObjectType
This attribute defines the BACnet Object type. The current supported types are listed in Section 4.9.

Column 4 – ObjectName
This identifier specifies the BACnet Object Name of the BACnet Object.

Column 5 – WritePriority
Here, the default write priority defined in the BACnet configuration file can be overwritten for each individual object. If not specified, the default write priority is used (cf. Section 4.3.3.1). As an alternative, a list of write priorities (separated by ",") can be specified, too. For each defined write priority, the server creates two additional server item – one for writing the datapoint value with the specified priority and one for resetting the priority again. Using this mechanism, multiple BACnet priorities can be used at the same time.

Column 6 – PollingInterval
If the object does not support Change of Value (COV) subscription, this parameter defines the interval (in ms) which is used to poll the object's present value. "'0"' deactivates polling for this object.

Column 7 – Persistent
This parameter specifies whether the value of the datapoint is persistent (restored from the database after server start up) or not.

Column 8 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not.

Column 9 – Synchronize
If this parameter is set to "T", the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

Column 10 – COV Type
This parameter specified the type of COV subscription that shall be used for the given object. "'Unconfirmed"' uses unconfirmed COV subscription, "'Confirmed"' uses confirmed COV subscription, and "'Disabled"' deactivates COV subscription at all. The default value (emtpy value) is "'Unconfirmed"'.

4.3.3.4. BACnet mapping definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.BACnet.MappingDefinitions.dat
```

The BACnet driver also implements the BACnet Server interface. This definition file is used to configure the mapping of Server Items to BACnet objects. Each valid configuration line represents one mapping and thus creates one BACnet object within the NETx BMS Server.

The structure of the BACnet mapping definition file is as follows:

```
' BACnet mapping definitions file
' ItemId;ObjectName;Relinquish-default;COV Increment;
'
NETx\XIO\KNX\BROADCAST\03/0/010;Light1;false
NETx\XIO\Modbus\Controller1\Input Registers\5000;EnergieValue1;0;0,5
```

Each line – except comment lines that start with ' – defines a single BACnet mapping and thus one create BACnet object.

Column 1 – ItemId
The ItemID of the Server Item that shall be mapped to a BACnet Object.

Column 2 – ObjectName
Here the BACnet object name for the newly created BACnet object has to be defined. Note that the BACnet object name must be unique within the server. The object name is optional – if not defined, the ItemID is used as object name.

Column 3 – Relinquish-default
This optional parameter defines the relinquish-default properties of the BACnet object. This value is used by the server all BACnet priorities are set to NULL.

Column 4 – COV Increment
COV Increment is an optional parameter which defines the minimum data change that is needed to send BACnet Change-of-Value notifications.

Column 5 – Object type
Here, the BACnet object type can be specified that is used to map the server item. If not specified or if the parameter

is left empty, the NETx BMS Server automatically detects the BACnet object type that fits best to the used Server Item.

Column 6 – Object ID
Here, the BACnet Object ID of the created BACnet object can be specified. The ID and the object type must be unique within the NETx BMS Server. If not specified or if the parameter is left empty, an auto-increment ID is used.

After start-up, the NETx BMS Server writes a list of all created BACnet into the following log file:

```
<WorkspaceDirectory>\LogFiles\xio.BACnet.ObjectList.log
```

The structure of this file is as follows:

```
' BACnet Object List Wed Jul 02 09:44:33 2014
' Object ID;Object Type;Object Name;Item ID
123123;Device;NETx BMS Server;
0;Binary Input;KNX Dimmer A - Status;NETx\XIO\KNX\192.168.1.36\05/0/001
```

Each line – except comment lines that start with ' – corresponds to one created BACnet object.

Column 1 – Object ID
This is the BACnet Object ID of the created BACnet object.

Column 2 – Object type
Here, the BACnet object type of the created object is listed.

Column 3 – ObjectName
This is the BACnet object name of the newly created BACnet object.

Column 4 – ItemId
This is the ItemID of the Server Item that is linked to the created BACnet Object.

4.3.4. JSON XIO Interface

JSON is a lean data exchange format that is also supported by the NETx BMS Server. Systems and their devices that exchange data via JSON objects can be integrated via the JSON interface. Within the current version, the metering data of ABB EQ meters can be collected.

4.3.4.1. JSON gateway definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.JSON.ABBEQGateways.dat
```

The gateway manager maintains the connections to the ABB JSON Gateway devices which are connected to the KNX system. The gateway definition table defines all the ABB JSON interfaces that are used within the project.

The structure of the gateway definition file is as follows:

```
'Syntax of the Gateway Definition Table:
'Gateway ID;IP Address;Port;User Name;Password;Polling Interval;Max. Requests
'
GW2;192.168.1.33;443;admin;admin;1000;1
```

Each line – except comment lines that start with ' – defines a single ABB JSON gateway.

Column 1 – Gateway ID
This column contains the ID of each the ABB JSON gateways. The ID is used to uniquely identify the gateway within the server. Therefore, it must be unique within the whole table and server. It is the reference to all those meters which are connected to this gateway.

Column 2 – IP Address
The content of this field defines the IP address of the gateway to communicate with.

Column 3 – Port
This attributes represents the port number which is used to connect to the gateway. The default value of the ABB interfaces is 443.

Column 4 – User Name
To get any information of the gateway it necessary to send user credentials to the interface. This field contains the user name.

Column 5 – Password
Column 5 contains the password of the user credentials for the communication with the ABB JSON gateway.

Column 6 – Polling Interval
The polling interval is set in milliseconds. It defines how often the state of the gateway is read. It has an indirect influence onto the polling intervals of the connected meters (c.f. 4.3.4.2).

Column 7 – Max. Requests
This defines the maximum number of requests sent to this gateway parallel at once.

4.3.4.2. JSON meter definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.JSON.ABBEQMeters.dat
```

Attached to each ABB JSON gateway are the meters. Their properties are stored and specified in this file. One meter can have more than one datapoint. The specifications of the meter will affect all datapoints related to this same meter.

The structure of the meter definition file is as follows:

```
'Syntax of the Meter Definition Table:
'Serial;Gateway ID;Type;Polling Interval;Description;Persistent;Historical;Synchronize
'
ABB01EM000001114;GW2;A44 352-100;5;my meter 1 on 33;F;T;F
ABB01EM000001015;GW2;A44 212-100;5;my meter 2 on 33;F;T;F
```

Each line – except comment lines that start with ' – defines a single ABB JSON meter.

Column 1 – Serial
The serial of a meter is unique and represents the meter behind the gateway. In order to gather information about the meter it needs to be uniquely addressed.

Column 2 – Gateway ID
This must be the same identifier like the one the system integrator set up in the gateway definition. This name defines how the NETx BMS Server will relate the meters to the gateways.

Column 3 – Type
The type of a meter tells the NETx BMS Server about the meter's structure of datapoints. The system integrator will be able to find this data in the web interface of the ABB JSON gateway.

Column 4 – Polling Interval
All values in this column are set in seconds. They stand in close relation to the polling interval of the gateway. A meter and its values will never be read more often than a gateway. Ideal is an integer multiple of the gateways polling interval. A polling interval of 1 second on the gateway and 5 seconds on the meter will cause the NETx BMS Server read the values of the meter every fifth time the gateway is read. Non whole numbered multiples of the gateway's polling interval will be truncated.

Example: Polling interval on gateway is 3000ms and polling interval on meter is 8s. This will lead to 8 divided by 3 which results into 2.667. This again will be truncated to 2. This results then into the fact that the meter will be read every second time the gateway will be polled.

Column 5 – Description
This attribute can be used to specify the gateway and its purpose closer to make it better readable for anyone working with it.

Column 6 – Persistent
This parameter specifies whether the value of the datapoints within the meter are persistent (saved to and restored from the database after server start up) or not.

Column 7 – Historical
This parameter specifies whether historical values of the datapoints within the meter are saved to the database or not.

Column 8 – Synchronize
If this parameter is set to "T", the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

4.3.4.3. JSON configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\xio.JSON.cfg
```

Within this configuration files, general configuration parameters of the JSON interface are set. Each parameter within the JSON configuration file is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

Parameter:	**XIO.JSON.ConnectionTimeout**
Scope:	IP address
Default value:	30
Unit:	seconds
Description:	Specifies the maximum time for the JSON gateway to respond network for communication before timeout.

4.3.5. SNMP XIO Interface

The SNMP interface provides the possibility to monitor devices in the network with enabled SNMP interface. SNMP stands for Simple Network Management Protocol and can be used to control networking devices. Furthermore the devices enable to read and write information on them. The information of a SNMP device is based on a so called SNMP Management Information Base (MIB) (Management information base) Tree, which can be browsed over object identifier so called SNMP Object Identifier (OID). This OID's are strings like 1.3.6.1.1.3.1.0, that represent the item path in the MIB tree. Each number specifies another OID value of another node in the tree and enables to navigate to the required value. The SNMP standard provides standardized function which allows to read and write information on this values. SNMP devices have additionally the possibility to send so called TRAP messages if specific events happen. This allows systems which receive this messages to react on this events with further actions. The NETx BMS Server allows the definition of items that contain the separated information which is delivered by the device when the TRAP message is delivered.

4.3.5.1. SNMP configuration

Location:

```
<WorkspaceDirectory>\ConfigFiles\xio.SNMP.cfg
```

Within this configuration files, general configuration parameters of the SNMP interface are set. Each parameter within the SNMP configuration file is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

Parameter:	**XIO:SNMP.DefaultCommunityString**
Scope:	string
Default value:	public
Unit:	None
Description:	This parameter is used to enable the authentication on the SNMP devices. If no string value is set by the user, the default community string is used.

Parameter:	**XIO:SNMP.PollingPath**
Scope:	string
Default value:	Polling
Unit:	None
Description:	The server items of the SNMP devices for polling MIB information are grouped in different paths. If a specific naming convention is required, the value can be set individually.

Parameter:	XIO:SNMP.UsePollingPath
Scope:	TRUE, FALSE
Default value:	FALSE
Unit:	None
Description:	If set to TRUE, the path provided in XIO:SNMP.PollingPath is used to create the polling items. Otherwise, the polling items are created in the device branch below the gateway data point.

Parameter:	XIO:SNMP.TrapPath
Scope:	string
Default value:	Traps
Unit:	None
Description:	Here, the path name is specified in which the trap items of a SNMP device are collected. If a specific name is required, the value can be set individually.

Parameter:	XIO:SNMP.DefaultGatewayItem
Scope:	string
Default value:	1.3.6.1.2.1.1.1.0
Unit:	None
Description:	The NETx BMS Server checks the availability of the SNMP device in the network on a standard item. This value specifies on which OID value the server should query the device.

Parameter:	XIO:SNMP.DefaultTimeout
Scope:	1000 - 60000
Default value:	3000
Unit:	None
Description:	Each time the NETx BMS Server queries the SNMP device for the configured items, the server waits for the configured time span for a response message.

4.3.5.2. SNMP device definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.DeviceDefinitions.dat
```

The SNMP devices definition specifies the communication components in the network. For each network component the device configuration needs to be specified. Based on the datapoint configuration which is discussed in the next section, the NETx BMS Server adds server items to the device which is represented in a separate directory. The parameter for the communication like IP address or port of the SNMP device are specified in this configuration.

The structure of the SNMP device definition file is as follows:

```
'SNMP device configuration file
'SNMP Server;Host IP;Port;Community string;Version code;Refresh interval;Max Request Items;Max Parallel cmds;Ignore
      Items Quality;Authentication name;
ServerRoomSensor;192.168.1.34;161;netxnetx;V2;5000;;;;;
TRAP_Wago1_22;192.168.1.22;161;public;V1;10000;;;;;
TRAP_Wago2_22;192.168.1.22;161;public;V2;10000;;;;;
TRAP_Wago3_22;192.168.1.22;161;public;V3;10000;;;;theuser;
```

Each line – except comment lines that start with ' – defines a single SNMP device.

Column 1 – SNMP Server
This parameter specifies the name of the SNMP device which represents the device on the NETx BMS Server and is important for the configuration of the data points.

Column 2 – Host IP
The parameter refers to the IP address of the SNMP device in the network.

Column 3 – Port
This attribute specifies the IP port of the SNMP device for the communication with the NETx BMS Server in the network. The default value is the "'161"'.

Column 4 – Community string
The authentication on a SNMP device requires a password which is called community string. The password can be individually set on each device, but the default value is "'public"'.

Column 5 – Version code
The SNMP standard has different versions with different functionality. The NETx SNMP Module supports Version 1

and Version 2 of the Standard. This parameter identifies which version is supported by the device and used during the communication. By default the version code is "'V1"', which stands for SNMP Version 1.

Column 6 – Refresh interval
This attribute specifies the time span in which the NETx BMS Server checks the availability of the SNMP device in the network. The default refresh interval is "'3000"' milliseconds.

Column 7 – Max Request Items
The number of polling items can be limited, so that the SNMP device is not overloaded. By default the item amount is restricted to "'10"' items.

Column 8 – Max Parallel cmds
Defines the number of requests which are used by the NETx BMS Server to poll the defined items. By default "'1"' request is used per device to query the data.

Column 9 – Ignore Items Quality
The parameter enables to reset item values on the NETx BMS Server, depending on the quality (status) of the SNMP device. The default value is "'F"' (false).

Column 10 – Authentication Name
SNMP version 3 requires allows the encryption of data which is transferred during the communication. The name refers to the configuration of the user in the SNMP user definition file which is decribed in section 4.3.5.4.

4.3.5.3. SNMP polling definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.PollingDefinition.dat
```

For the configuration of SNMP polling items, the definition file needs to be used. The NETx BMS Server polls the information of the SNMP device. The received data is finally represented over the server items of the NETx BMS Server. This allows to monitor each single MIB item of the SNMP device.

The structure of the SNMP polling definition file is as follows:

```
' SNMP polling configuration file
'Server Name;Object ID;Name;Description;Data type;Access mode;Polling interval;Persistent;Historical;Synchronize;
TRAP_Wago1_22;1.3.6.1.2.1.1.1.0;1.3.6.1.2.1.1.1.0;Desc1;;RW;5000;;;;
TRAP_Wago1_22;1.3.6.1.2.1.1.2.0;1.3.6.1.2.1.1.2.0;Desc2;;RW;5000;;;;
TRAP_Wago1_22;1.3.6.1.2.1.1.3.0;1.3.6.1.2.1.1.3.0;Desc3;;RW;5000;;;;
TRAP_Wago1_22;1.3.6.1.2.1.1.4.0;1.3.6.1.2.1.1.4.0;Desc2;;RW;5000;;;;
TRAP_Wago2_22;1.3.6.1.2.1.1.1.0;1.3.6.1.2.1.1.1.0;Desc2;;RW;5000;;;;
TRAP_Wago2_22;1.3.6.1.2.1.1.2.0;1.3.6.1.2.1.1.2.0;Desc2;;RW;5000;;;;
TRAP_Wago2_22;1.3.6.1.2.1.1.3.0;1.3.6.1.2.1.1.3.0;Desc2;;RW;5000;;;;
TRAP_Wago2_22;1.3.6.1.2.1.1.4.0;1.3.6.1.2.1.1.4.0;Desc2;;RW;5000;;;;
TRAP_Wago3_22;1.3.6.1.2.1.1.1.0;1.3.6.1.2.1.1.1.0;Desc2;;RW;5000;;;;
TRAP_Wago3_22;1.3.6.1.2.1.1.2.0;1.3.6.1.2.1.1.2.0;Desc2;;RW;5000;;;;
TRAP_Wago3_22;1.3.6.1.2.1.1.3.0;1.3.6.1.2.1.1.3.0;Desc2;;RW;5000;;;;
TRAP_Wago3_22;1.3.6.1.2.1.1.4.0;1.3.6.1.2.1.1.4.0;Desc2;;RW;5000;;;;
ServerRoomSensor;1.3.6.1.2.1.1.1.0;System Description;Desc2;;RW;5000;;;;
ServerRoomSensor;1.3.6.1.2.1.1.2.0;System ObjectID;Desc2;;RW;5000;;;;
ServerRoomSensor;1.3.6.1.2.1.1.3.0;System UpTime;Desc2;;RW;5000;;;;
ServerRoomSensor;1.3.6.1.2.1.1.4.0;System Contact;Desc2;;RW;5000;;;;
```

Each line – except comment lines that start with ' – defines a single SNMP item value which is represented over one or more NETx BMS Server items.

Column 1 – Server Name
The Server Name value references to the device configuration in which this value is used to define the global device parameter for the communication. For a successful configuration process the module needs to find the device in the device configuration file.

Column 2 – Object ID
This parameter refers to the SNMP OID values of the MIB tree. This value has to be unique for each device in the configuration and needs to be set.

Column 3 – Name
This attribute defines the name of the NETx BMS Server server item. If the value is not set in the configuration file, the OID attribute is used to define a name for the server item.

Column 4 – Description
The parameter enables the user to add a description for the data point.

Column 5 – Data type
The parameter enables to define the data type of the SNMP device item, which is represented on the server. By default the data type is a "'string"' value.

Column 6 – Access mode
This parameters specifies the access mode of the data point on the server. The mode can either be Read, Write or ReadWrite depending on the SNMP device item. The default value is Read.

Column 7 – Polling interval
The polling interval specifies the time period, after which the NETx BMS Server checks again for the item. The default value is "'5000"' milliseconds until the next period.

Column 8 – Persistent
This parameter specifies whether the value of the data point is persistent (restored from the database after server start up) or not. The default value is "'F"' (false).

Column 9 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not. The default value is "'F"' (false).

Column 10 – Synchronize
If this parameter is set to "T", the value is synchronized between the main and backup server (if present). This will work only if synchronization is enabled in the NMESH configuration of the NETx BMS Server. The default value is "'T"' (true).

4.3.5.4. SNMP user definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.UserDefinitions.dat
```

The SNMP Module provides the transfer of encrypted data between the device and the NETx BMS Server. The defined user have to be added to the SNMP device definition file, to allow the successful decryption of data from the device.

The structure of the SNMP user definition file is as follows:

```
'SNMP user configuration file
'Authentication name; Authentication Protocol; Authentication key; Privacy Protocol; Privacy password
user;NONE;NoAuthenticationKey;NONE;Noprivacykey;
myuser;MD5;mypassword;DES;myotherpassword;
theuser;SHA1;AuthenticationKey;AES;PrivacyKey;
```

Each line – except comment lines that start with ' – defines a single SNMP user definition.

Column 1 – Authentication name
This parameter specifies the name of the SNMP user which is used for the reference in the SNMP device definition file as well as for the authentication on the device.

Column 2 – Authentication protocol
This parameter specifies the protocol which is used for the authentication procotol. The parameter allows the selection of the options MD5, SHA1 or the definition of no protocol.

Column 3 – Authentication key
The authentication key is the password string which used for the authentication of the user on the system.

Column 4 – Privacy protocol
The privacy protocol is used to encrypt the data of the SNMP device. The parameter provides the options AES, DES and None.

Column 5 – Privacy password
The privacy password represents the password string which is used for the decryption of the SNMP data.

4.3.5.5. SNMP trap definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.TrapDefinitions.dat
```

Defines the origin OID of the SNMP trap message which is represented over a NETx BMS Server item. If further information of the trap message should be represented in the Item Tree of the server, the trap variable definition file has to be configured accordingly.

The structure of the SNMP trap definition file is as follows:

```
' SNMP trap definition file
'Server Name;Trap Object ID;Name;Description;Data type;Persistent;Historical;Synchronize;
TRAP_Wago1_22;1.3.6.1.6.3.1.1.5;ColdStart;ColdStart;;;;
TRAP_Wago2_22;1.3.6.1.6.3.1.1.5.1;ColdStart;ColdStart;;;;
TRAP_Wago2_22;1.3.6.1.6.3.1.1.5.4;Linkup;Linkup;;;;
TRAP_Wago3_22;1.3.6.1.6.3.1.1.5;ColdStart;ColdStart;;;;
```

Each line – except comment lines that start with ' – defines a single SNMP item value which is represented over one or more NETx BMS Server items.

Column 1 – Server Name
The Server Name value references to the device configuration in which this value is used to define the global device parameter for the communication. For a successful configuration process the module needs to find the device in the device configuration file.

Column 2 – Trap Object ID
This parameter refers to the SNMP OID values of the MIB tree. This value has to be unique for each device in the configuration and needs to be set.

Column 3 – Name
This attribute defines the name of the NETx BMS Server server item. If the value is not set in the configuration file, the OID attribute is used to define a name for the server item.

Column 4 – Description
The parameter enables the user to add a description for the data point.

Column 5 – Data type
The parameter enables to define the data type of the SNMP device item, which is represented on the server. By default the data type is a "'string"' value.

Column 6 – Persistent
This parameter specifies whether the value of the data point is persistent (restored from the database after server start up) or not. The default value is "'F"' (false).

Column 7 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not. The default value is "'F"' (false).

Column 8 – Synchronize
If this parameter is set to “T”, the value is synchronized between the main and backup server (if present). This will work only if synchronization is enabled in the NMESH configuration of the NETx BMS Server. The default value is "'T"' (true).

4.3.5.6. SNMP trap variable definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.TrapVariableDefinitions.dat
```

In combination with the trap definitions file, the trap variable definition enables to represent all information of a SNMP trap message over NETx BMS Server items.

The structure of the SNMP trap variable definition file is as follows:

```
' SNMP trap definition file
'Server Name;Trap Object ID;Variable Object ID;Name;Description;Data type;Persistent;Historical;Synchronize;
TRAP_Wago2_22;1.3.6.1.6.3.1.1.5.4;1.3.6.1.2.1.2.2.1.1.1;Var1;Desc1;;;;;
TRAP_Wago2_22;1.3.6.1.6.3.1.1.5.4;1.3.6.1.2.1.2.2.1.7.1;Var2;Desc2;;;;;
TRAP_Wago2_22;1.3.6.1.6.3.1.1.5.4;1.3.6.1.2.1.2.2.1.8.1;Var3;Desc3;;;;;
TRAP_Wago3_22;1.3.6.1.6.3.1.1.5;1.3.6.1.6.3.1.1.5.5.5;ColdStart;ColdStart1;;;;;
TRAP_Wago3_22;1.3.6.1.6.3.1.1.5.4.1;1.3.6.1.2.1.2.2.1.1.1;Var54;Desc1;;;;;
TRAP_Wago3_22;1.3.6.1.6.3.1.1.5.4.2;1.3.6.1.2.1.2.2.1.7.1;Var55;Desc2;;;;;
TRAP_Wago3_22;1.3.6.1.6.3.1.1.5.4.3;1.3.6.1.2.1.2.2.1.8.1;Var56;Desc3;;;;;
'Deb1;1.3.6.1.6.3.1.1.5.1;1.3.6.1.6.3.1.1.5.5.5;ColdStartVar1;ColdStart3;;;;
'TRAP_Wago1_22;1.3.6.1.6.3.1.1.5;1.3.6.1.6.3.1.1.5.5.5;ColdStart;ColdStart1;;;;
```

Each line – except comment lines that start with ’ – defines a single SNMP item value which is represented over one or more NETx BMS Server items.

Column 1 – Server Name
The Server Name value references to the device configuration in which this value is used to define the global device parameter for the communication. For a successful configuration process the module needs to find the device in the device configuration file.

Column 2 – Trap Object ID
This parameter refers to the SNMP OID values of the MIB tree. This value has to be unique for each device in the configuration and needs to be set.

Column 3 – Variable Object ID
This parameter refers to the SNMP trap message values. This value has to be unique for each trap message in the configuration and needs to be set.

Column 4 – Name
This attribute defines the name of the NETx BMS Server server item. If the value is not set in the configuration file, the OID attribute is used to define a name for the server item.

Column 5 – Description
The parameter enables the user to add a description for the data point.

Column 6 – Data type
The parameter enables to define the data type of the SNMP device item, which is represented on the server. By default the data type is a "‘string"’ value.

Column 7 – Persistent
This parameter specifies whether the value of the data point is persistent (restored from the database after server start up) or not. The default value is "‘F"’ (false).

Column 8 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not. The default value is "‘F"’ (false).

Column 9 – Synchronize
If this parameter is set to “T”, the value is synchronized between the main and backup server (if present). This will work only if synchronization is enabled in the NMESH configuration of the NETx BMS Server. The default value is "‘T"’ (true).

4.3.5.7. SNMP trap listener definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.SNMP.TrapListenerDefinitions.dat
```

The definition of Trap listener enables to specify ports, where the NETx BMS Server listens for incomming SNMP messages.

The structure of the SNMP trap listener definition file is as follows:

```
’SNMP trap listener file
’IP address; Port;
ANY;161;
192.168.1.1;162;
```

Each line – except comment lines that start with ’ – defines a single SNMP trap listener definition.

Column 1 – IP Address
Specifies the IP address of the SNMP device, which sends the trap messages to the NETx BMS Server. If multiple different addresses are required, the keyword "‘ANY"’ has to be used, to define that any IP Address is allowed to send messages.

Column 2 – Port
Defines the IP Port the NETx BMS Server has to listen for incoming trap messages.

4.4. Server modules

The NETx BMS Server provides so called server modules. These modules can be used to implement functionality that is necessary by special sub systems of the building automation system. Acting on the datapoints that are handled within the server, these modules provide a specific view of the corresponding datapoints that can be used by BMS Clients to fulfill the requirements of these special sub system. A typical example is the Metering module that provide reporting and analysis functionality of smart meters. The remainder of section describes the different server modules that are supported within the current version of the NETx BMS Server.

4.4.1. Cluster module

The cluster module of the NETx BMS Server can be used to integrate datapoints from foreign OPC DA and/or remote NETx BMS Servers. The cluster module is also used to interconnect multiple NETx BMS Servers in a hierarchy. This so called clustering provides the opportunity to exchange and forward datapoints of different NETx BMS Servers. A typical example would be a main server that integrates datapoints from different, independent sub-servers.

4.4.1.1. Cluster server definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaClusterServer.40.dat
```

Using the cluster server definition file, the different OPC DA and NETx BMS Server s that shall be integrated are specified.

The structure of the cluster server definition file is as follows:

```
' Cluster Server Definition File
'
' ServerName;ServerType;IPAddress;CommunicationParameter;Description
OPCX;DA;192.168.0.101;Opc.Vendor.X;OPC Server of vendor X
NETxOPC;DA;;NETxKNX.OPC.Server.3.5; NETx KNX OPC Server 3.5
NETxSubBMS1;VNET;192.168.0.102;4530;Sub Server 1
NETxSubBMS2;VNET;192.168.0.103;4530;Sub Server 2
```

Each line – except comment lines that start with ' – specifies one OPC DA or one NETx BMS Server.

Column 1 – ServerName
The name of the OPC DA or NETx BMS Server. For all servers, this name has to be unique since it is used as unique identifier.

Column 2 – ServerType
Here the type of the server has to be specified. Valid types are “DA” for OPC DA 2.05 Servers and “VNET” for NETx BMS Servers.

Column 3 – IPAddress
This parameter defines the host name or the IP address of the server. If this parameter is empty, “localhost” is used.

Column 4 – CommunicationParameter
Here the communication parameter for the corresponding server connection has to be configured. For OPC DA servers (ServerType “DA”), the OPC Program ID has to be defined (e.g. for NETx KNX OPC Servers: “NETxKNX.OPC.Server.3.5”). For NETx BMS Servers (ServerType “VNET”), the VNET port (default 4530) has to be specified.

Column 5 – Description
This parameter specifies the description of the server.

4.4.1.2. Cluster item definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaClusterItem.40.dat
```

Using the cluster item definition file, the different datapoints that shall be integrated into the NETx BMS Server are specified.

The structure of the cluster item definition file is as follows:

```
' Cluster Item Definition File
'
' ServerName;ItemID;Description;ReadInterval;Autorefresh;Persistent;Historical;Backup
OPCX;OPCItem1;\Path\To\Item;Foreign OPC Item 1;1000;F;F;F;F
NETxOPC;KNXItem1;\NETxKNX\BROADCAST\04/0/001;0;T;F;F;F
NETxSubBMS1;Item1;NETx\XIO\Modbus\Controller1\Input Registers\5000;0;T
NETxSubBMS2;Item2;NETx\XIO\BACnet\Device0\BINARY0\BINARY0;0;T
```

Each line – except comment lines that start with ' – defines a single datapoint that shall be integrated into the NETx BMS Server.

Column 1 – ServerName
This is the name of the server that provides the datapoint. It has to match the name that is used in the cluster server definition file (cf. Section 4.4.1.1).

Column 2 – ItemID
Here the ItemID that is used within the foreign server to identify the datapoint has to be specified.

Column 3 – Description
This parameter specifies the description of the datapoint.

Column 4 – ReadInterval
Using this parameter, the polling interval that is used to cyclically read the datapoint within the remote server can be specified. If it is set to "0", cyclically reading the value is disabled.

Column 5 – Autorefresh
If this parameter is set to "T", the NETx BMS Server subscribes to the datapoint within the remote server. Using subscription, data changes are automatically pushed from the remote server to the NETx BMS Server. If the remote server is a NETx BMS Server (ServerType is set to "VNET"), then subscription is always used.

Column 6 – Persistent
This parameter specifies whether the value of the datapoint is persistent (restored from the database after server start up) or not.

Column 7 – Historical
This parameter specifies whether historical values of the datapoint are stored within the database or not.

Column 8 – Synchronize
If this parameter is set to "T", the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

4.4.2. Metering module

The Metering module is used to collect data from smart metering devices (e.g. water meters, electricity meters, . . .). Based on the collected data and on user-defined cost rates, cost calculations are performed and stored within the data model of the server. This calculated data can then be accessed by any client to perform reporting and trending tasks.

The configuration of the smart metering devices that are used in the Metering is done in the Metering module configuration file.

Location:

```
<WorkspaceDirectory>\DataFiles\xmo.MaRS.Meter.dat
```

The structure of the Metering module definition file is as follows:

```
'Syntax of the Metering Module configuration file:
'MeterName;Description;Resource;Input ItemID;Input Type;Meter interval;Scale factor
Meter1;Water meter building A;Water;NETx\XIO\KNX\BROADCAST\00/0/006;Increasing;5;1
Meter2;Electricity meter building A;Electricity;NETx\XIO\BACnet\BACnetController1\Accumulator1;Current;15;10
Meter3;Another meter;EnergyResource1;NETx\VAR\Real\Item5;Impulse;15;5
```

Each line – except comment lines that start with ' – defines a single Metering device.

Column 1 – Meter Name

The name of the virtual meter. For all Metering devices, this name has to be unique since it is used as unique identifier for every single meter.

Column 2 – Description
This attribute can be used to specify a human-readable text that further describes the meter (can be any user-defined string).

Column 3 – Resource
This parameter defines the type of resource that is monitored by the meter. Since this resource name is only used within the Server Item Tree to organize the meters in hierarchical view, any user-defined string can be used. Meters that shall be arranged within the same sub tree must have the same resource name.

Column 4 – Input Item ID
This parameter specify the Server Item ID of the datapoint that is the source of the meter. The source of the meter can be any Server Item (even custom items that only available within the server).

Column 5 – Input Type
This parameter specifies the meter type. In the current version, the following types are supported:

- "Increasing" – The difference between the start value and stop value of the source datapoint within the current interval (ref. column 6) is set as the metering value.
- "Impulse" – The amount of value changes within the current interval are counted (independent of the value of the source datapoint)
- "Current" – The actual value of the source datapoint is added to the metering value if the value changes.

Example: Consider, for example, the source datapoint has a value of "10" and the metering value is "100". Afterwards, the value of the source data point is set to "12". After the next metering interval, metering value is updated. If the meter is of type "Increasing", the new metering value is "102". If the meter is of type "Impulse", the new metering value is "101". If the meter is of type "Current", the new metering value is "112". Now, consider, for instance, the value source datapoint is set to "14" and then to "16". After the next metering interval, the new metering value of the "Increasing" meter is "106", the new metering value of the "Impulse" meter is "103", and the new metering value of the "Current" meter is "142".

Column 6 – Interval
This attributed defines the time interval in minutes that is used to update the metering value within the Metering module. Possible values are "5", "10", "15", "20", and "30". Note that this value does not influence the update interval of the source datapoint – it is only used to update the metering value within the Metering database.

Column 7 – Scale Factor
Using this parameter, it is possible to perform a linear scaling of the metering value.

4.4.3. Fidelio module

Hotel management systems often use their own protocols for communication. Therefore, the NETx BMS Server offers an optional interface to Fidelio/Opera.

4.4.3.1. Driver Configuration

The configuration of the Fidelio interface is done in the Fidelio module configuration file.

Location:

```
<WorkspaceDirectory>\ConfigFiles\nxapiFIDELIO.cfg
```

Within this configuration, all Fidelio settings are specified. Each parameter is identified with a name. The name is case-insensitive (i.e. capitalization is not considered). The scope indicates the allowed values of the parameter. The default value specifies the values that is used when no value is set by the user.

The parameters which can be defined in this file are divided into the following two groups:

- FIDELIO
- SYS

FIDELIO-Parameters

Parameter:	**FIDELIO.IPAddress**
Scope:	###.###.###.###
Default value:	None
Unit:	None
Description:	This defines the IP address of the FIDELIO or OPERA server.

Parameter:	**FIDELIO.Port**
Scope:	### (1-65535)
Default value:	5018
Unit:	None
Description:	This defines the port number of the FIDELIO or OPERA server.

Parameter:	**FIDELIO.Logging**
Scope:	ON/OFF
Default value:	ON
Unit:	None
Description:	This enables the logging of the data telegrams from the FIDELIO or OPERA server.

Parameter:	**FIDELIO.LRC**
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	This turns on or off LRC for the data telegrams.

Parameter:	**FIDELIO.UpdateOnReconnect**
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	If set data from FIDELIO or OPERA server are called to update the information inside the NETx BMS Server when the connection to the FIDELIO or OPERA server is reestablished.

SYSTEM-Parameters

Parameter:	**SYS.RefreshInterval**
Scope:	###
Default value:	20
Unit:	seconds
Description:	This defines the time between one call for data and the next one to the FIDELIO or OPERA server.

4.4.3.2. Room Definitions

Location:

```
<WorkspaceDirectory>\DataFiles\xio.JSON.ABBEQGateways.dat
```

The room definition defines the structure of the FIDELIO tree. Each room has the same structure inside. Thus only the rooms need to be defined.

The structure of the FIDELIO room definition file is as follows:

```
'RoomNumber;BuildingName;FloorName;Description;Number
1;House1;Floor1;House 1, Floor 1, Room 1;0
2;House1;Floor1;House 1, Floor 1, Room 2;1
11;House1;Floor2;House 1, Floor 2, Room 1;2
12;House1;Floor2;House 1, Floor 2, Room 2;3
```

Column 1 – Room Number
This field is the real room number of the room described in this line.

Column 2 – Building Name
The building name creates the first sub-folder inside the NXA_FIDELIO branch under NETx\API. All rooms defined which have the same building name defined will appear under one and the same branch.

Remark: “House 1” does not equal “HOUSE 1”

Column 3 – Floor Name
The level name (or floor name) creates the second hierarchy below the different buildings. All rooms defined which have the same level name and building name defined will appear under one and the same branch.

Remark: “Floor 1” does not equal “FLOOR 1”

Column 4 – Description
Place any description for the room here.

Column 5 – Number
This field contains the ID of the room in numeric form. It must be unique.

4.5. Server extensions

The NETx BMS Server provides so called server extensions.

4.5.1. XCommand event definitions

Location:

```
<WorkspaceDirectory>\EventFiles\nxaXLogicEvents.dat
```

The definition of XCommands provide the possible linkage of logical functions between multiple different server items. The defined XCommands can be executed by defining events which react on input value of the involved server items.

The structure of the XCommand definitions file is as follows:

```
' Name;VariableGroup;Type;Options;XCommand;Inputs;Outputs;Parameters
OR Example;;ON_INPUT;;Or;NETx\VAR\Boolean\Item1,NETx\VAR\Boolean\Item2,NETx\VAR\Boolean\Item3;NETx\VAR\Boolean\Item4;
```

Each line – except comment lines that start with ’ – defines a single task.

Column 1 – Name
This parameter defines the name of the XCommand instance. The name must be unique.

Column 2 – Variable group
This parameter defines the scope where the used internal variables of XCommand are valid. All XCommand instances that are using the same variable group are accessing the same internal variables. This means that all XCommand instances are sharing the same internal variables. If left empty, the instance name (column 1) is used as variable group.

Column 3 – Type
This attribute defines when the XCommand shall be triggered. “ON_START” means that the XCommand is executed once when the NETx BMS Server starts. “ON_TIMER” triggers the XCommand cyclically (the interval in seconds can be specified in “Options” column). "‘ON_INPUT"’ executes the XCommand whenever one input item changes its value. “ON_TIMER_AND_INPUT” is used for XCommands that shall be triggered cyclically and whenever one input item changes. “ON_STOP” means that the XCommand is executed once when the NETx BMS Server stops.

Column 4 – Options
If “ON_TIMER” is selected, the time interval (in seconds) can be specified here.

Column 5 – XCommand
The parameter defines the XCommand that shall be executed. To assist in selecting the inputs, outputs, and parameters, a dialog is available which can be openend via the “...” button.

Column 6 – Inputs
This parameter defines the inputs of the XCommand. If the XCommand is selected via the dialog, this parameter is filled automatically.

Column 7 – Outputs
This parameter defines the outputs of the XCommand. If the XCommand is selected via the dialog, this parameter is filled automatically.

Column 8 – Parameters
This parameter defines the parameters of the XCommand. If the XCommand is selected via the dialog, this parameter is filled automatically.

4.5.2. Task definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaTaskDefinitions.35.dat
```

Task definitions can be used to react on value changes of Server Items by initiating actions that may change the value of other item(s) or execute other tasks. Using this mechanism, it is possible to relate two Server Items to each other – even when they are located at networks that use different communication protocols. For example, it is possible to link a KNX datapoint to a BACnet datapoint. In addition to a simple mapping of datapoints, it is also possible to process and modify datapoints. By defining LUA script functions that are executed when a task is triggered, any control functionality can be implemented that is performed whenever the specified task condition is triggered.

The structure of the task definitions file is as follows:

```
' SourceItem;DestinationItem;OnReceive;OnSend;OnSetValue;Delay;Command;Parameter
NETx\XIO\KNX\BROADCAST\00/7/000;NETx\XIO\KNX\BROADCAST\00/7/001;T;T;T;0;READ;
NETx\XIO\BACnet\BACnetDevice1\Object1;NETx\XIO\KNX\BROADCAST\00/7/002;T;T;T;0;WRITE;
NETx\XIO\Modbus\ModBusDevice1\coil\1;NETx\XIO\KNX\BROADCAST\00/7/003;T;T;T;5000;SET;
MyCustomTree\MyCustomItem1;;T;T;T;1000;SCRIPT;MyLUAFunc()
```

Each line – except comment lines that start with ' – defines a single task.

Column 1 – Source Item
This parameter defines the Item ID of the source Server Item that triggers the task.

Column 2 – Destination Item
This attribute defines the Item ID of the destination Server Item that may be changed by the task.

Column 3 – OnReceive
If this parameter is set to T, the task is triggered when a read telegram is received at the specified source Server Item. Otherwise, the task is not triggered.

Column 4 – OnSent
If this parameter is set to T, the task is triggered when a write telegram is received at the specified source Server Item. Otherwise, the task is not triggered.

Column 5 – OnSetValue
If this parameter is set to T, the task is triggered when the value of the specified source Server Item is set (e.g. by a BMS Client). Otherwise, the task is not triggered.

Column 6 – Delay in ms
Here, a delay (in milliseconds) for executing the task can be specified.

Column 7 – Command
This parameter is used to specify the action that is performed when the task is triggered. Currently, the following actions are supported:

- WRITE: the server will generate also a write telegram for specified destination Server Item.
- READ: the server will generate also a read telegram for specified destination Server Item.
- SET: the server will set the value of the specified destination Server Item.
- SCRIPT: the server will execute a LUA script function. The name of the function is specified within the column “Parameters”.

Column 8 – Parameters
It is used to define the LUA script, which should be executed.

4.5.3. Holiday definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaHolidays.40.dat
```

Holidays are specific to each country and often enough even to fractions in there. Therefore the possibility to administrate them was given to the system integrator. All specified holidays are stored in this one file. Holidays are important parameters to the server side calendar. Recurring calendar events can explicitly include or exclude holidays.

The structure of the holiday definition file is as follows:

```
'Syntax of the Holiday Definition Table:
'Day;Month;Year;Name
1;1;2000;New Years Day
```

Each line – except comment lines that start with ' – defines a single holiday.

Column 1 – Day
This field defines the day of the holiday.

Column 2 – Month
The content of this column is the month of the specified holiday.

Column 3 – Year
The year of the holiday is saved in this field of the table.

Column 4 – Name
To name a defined holiday a string needs to be put into this column.

4.5.4. Aliases definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaAliases.40.dat
```

Aliases can be used to give any server item a user-defined name. Aliases can be seen as a short-cut to the original server item. Therefore, an alias is not a copy of the source item – it is just alternative way to access an item. Aliases can be used to give physical datapoints a more descriptive name. Furthermore, aliases can also be arranged in a user-defined hierarchy.

The structure of the alias definition file is as follows:

```
' Aliases Definition File
'
' SourceItemID;Name;Path
NETx\XIO\KNX\BROADCAST\03/0/002;LightOff;BuildingA\Floor1\
NETx\XIO\BACnet\Device1\ANALOG_VALUE_0\ANALOG_VALUE_0 - PRIO 15;HeatingSetpoint;BuildingA\Floor1\RoomA
```

Each line – except comment lines that start with ' – defines a single alias.

Column 1 – SourceItemID
This is the source ItemID of the alias.

Column 2 – Name
Here the name of the alias has to be specified.

Column 3 – Path
Optionally, the alias can be arranged in a hierarchy which can be defined here. The different levels of the hierarchy must be delimited with "\".

4.5.5. Virtual item definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaVirtualItems.40.dat
```

Virtual items are used like any other server item except for one detail; they are not connected to any real datapoint on bus level. They can be used to store data. They may be connected to a any other server item.

The structure of the virtual item definition file is as follows:

```
'Syntax of the Virtual Item Definition Table:
'ID;Path;Description;Data Type;Access Rights;Persistent;Historical;Synchronized;Link Item ID;Script For OnReceive;
      Script For OnSend;Script For OnSet
GlobalOff;BuildingA\Floor1\RoomA;Virtual Item for global off;BOOL;RW;;;;;;;;;
```

Each line – except comment lines that start with ' – defines a single virtual item.

Column 1 – ID
This field is naming the virtual item. Together with the path it must be unique.

Column 2 – Path
The string within this field determines the path of the virtual item. Delimiter between sections of the path is the backslash (“
”).

Column 3 – Description
Set the description for the virtual item here. It will be displayed in the item tree.

Column 4 – Data Type
Possible data types are BOOL, STRING, FLOAT, DOUBLE, INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, and UINT64.

Column 5 – Access Rights
Shall the virtual item be writable or is it read only?

Column 6 – Persistent
This parameter specifies whether the value of the virtual item is persistent (restored from the database after server start up) or not.

Column 7 – Historical
This parameter specifies whether historical values of the virtual item are stored within the database or not.

Column 8 – Synchronize
If this parameter is set to “T”, the value is synchronized between the main and backup server (if present). This will work only if synchronize is enabled in the NMESH configuration of the NETx BMS Server.

Column 9 – Link Item ID
If a connection to another item is necessary the item ID of this one has to be placed here.

Column 10 – Script For OnReceive
Implement any LUA script here which shall be called at the OnReceived event.

Column 11 – Script For OnSend
Implement any LUA script here which shall be called at the OnSent event.

Column 12 – Script For OnSet
Implement any LUA script here which shall be called at the OnSet event.

Column 13 – Extended Data 1
Internal use only.

Column 14 – Extended Data 1
Internal use only.

Column 15 – Extended Data 1
Internal use only.

4.5.6. Virtual link definitions

Location:

```
<WorkspaceDirectory>\DataFiles\nxaVLinks.40.dat
```

Virtual links can connect all kinds of items. In example an item from KNX gets a new value, then this value can be forwarded to a Modbus or BACnet item or vice versa. Virtual items can be linked as well.

The structure of the virtual link definition file is as follows:

```
'Syntax of the Virtual Link Definition Table:
'Source Item ID;Destination Item ID
NETx\XIO\KNX\192.168.1.7\03.0.010;NETx\XIO\Modbus\MODBUS_01\Coils\12420
```

Each line – except comment lines that start with ' – defines a single virtual link.

Column 1 – Source Item ID
This field contains the server item from which the value is taken.

Column 2 – Destination Item ID
To this server item the source value will be written.

4.6. LUA scripting

Whenever control functionality cannot be added by using the mechanisms mentioned before (e.g. Task definitions), the scripting engine of the NETx BMS Server can be used. This scripting engine provides the opportunity to execute LUA scripts that implement the required control functionality. Using LUA scripts, the user has full access to the functionality of the server – nearly any control functionality can be implemented this way. Moreover, most of the provided functionality can also be changed during runtime and so LUA scripts provides one of the most flexible way to extend the functionality of the NETx BMS Server system.

LUA itself is an open source scripting language. There are two main reasons why LUA was chosen as scripting language. First, LUA is very fast and so LUA scripts scale to large systems which is of utmost importance in the building automation domain – LUA scripts can handle thousands of datapoints. Second, due to the simple syntax of LUA, LUA scripting can easily be learned – even for beginners it is easy to write their own LUA scripts.

> ! However, as it is common for each programming language, some basic training is necessary to write LUA scripts. For the general syntax of the LUA scripting language please refer to www.lua.org. Here, tutorials and how-to's that can be used as a good starting point for beginners can be found.

LUA scripts are stored in normal text files that use the extension ".lua". Within the NETx BMS Server, all LUA scripts have to be stored within the script directory of the current workspace:

```
<WorkspaceDirectory>\ScriptFiles\
```

While it is possible to write LUA scripts in any file within this directory, the following script is the default LUA script that is loaded by the server at start up:

```
<WorkspaceDirectory>\ScriptFiles\nxaDefinitions.lua
```

To avoid that each user must start from scratch when he or she wants to implement some control functionality with LUA scripts, the so called NXA LUA library is already integrated within the NETx BMS Server. Using the included functions and event callbacks, the user can access the internal functionality of the server. For example, using appropriate LUA functions, Server Items and Task definitions can manipulated even during runtime. In general, the following LUA functionality is provided by the NXA LUA library:

- NXA standard functions: These functions can be used at anytime during runtime. Typical examples are functions that change the value of Server Items or functions that can be invoked to retrieve the current workspace name (cf. Section 4.6.1).
- NXA functions for task handling: These functions are used in combination with tasks (cf. Section 4.6.2).
- NXA functions for KNX In Out Conversion: These functions are used for KNX items, which needs to be converted or only certain bits are used (cf. Section 4.6.3).
- NXA functions for creating custom Server Items: These functions can be used to define user-specific custom items and hierarchies within the Server Item Tree (cf. Section 4.6.7).
- NXA event callbacks: In addition, the NXA library also provides event callbacks that invoked by the server at a specific point in time. A typical example is the "OnInitEvent" callback that is invoked during server start up (cf. Section 4.6.10).
- NXA constants: Furthermore, some constants are also defined in the library that assist the user in programming the required functionality (cf. Section 4.6.11).

In the remainder of this section, these different functions, callbacks, and constants are explained in more detail. Furthermore, some additional information that helps to implement LUA scripts is also provided at the end of this section (cf. Section 4.6.12).

4.6.1. NXA standard functions

NXA standard functions provide basic functionality that can be used in any LUA function at anytime during runtime.

nxa.IsInitialized
This function can be used to determine whether the server is initialized or not.

Returns:

- bool – true when server is initialized, otherwise false.

nxa.IsRunning
This function can be used to determine whether the server is running or not.

Returns:

- bool – true when server is running, otherwise false.

nxa.IsSimulation
This function can be used to determine whether the server is in simulation mode or not.

Returns:

- bool – true when server is in simulation mode, otherwise false.

nxa.IsActiveServer
This function can be used to determine whether the server is active or not.

Returns:

- bool – true when server is active, otherwise false.

nxa.IsMainServer
This function can be used to determine whether the server is defined as main server or not.

Returns:

- bool – true when server is defined as main server, otherwise false.

nxa.IsBackupServer
This function can be used to determine whether the server is defined as backup server or not.

Returns:

- bool – true when server is defined as backup server, otherwise false.

nxa.WorkspaceName
This function returns the current name of the workspace.

Returns:

- string – workspace name

nxa.RootPath
This function returns the path of the server installation.

Returns:

- string – root program path of the installed server application

nxa.WorkspacePath
This function returns the path of the current workspace.

Returns:

- string – workspace path

nxa.ScriptFilesPath
This function returns the path to the script file directory of the current workspace.

Returns:

- string – script file path

nxa.ConfigFilesPath
This function returns the path to the config file directory of the current workspace.

Returns:

- string – data file path

nxa.DataFilesPath
This function returns the path to the data file directory of the current workspace.

Returns:

- string – data file path

nxa.LogFilesPath
This function returns the path to the log file directory of the current workspace.

Returns:

- string – log file path

nxa.ProjectFilesPath
This function returns the path to the project file directory of the current workspace that contains the NETx BMS Client files (*.vxf).

Returns:

- string – project files path

nxa.EventFilesPath
This function returns the path to the event file directory of the current workspace.

Returns:

- string – event file path

nxa.LogInfo
This function generates an info message that is logged to the system log file.

Parameters:

- string – text of the info message

nxa.LogWarning
This function generates a warning message that is logged to the system log file.

Parameters:

- string – text of the warning message

nxa.LogError
This function generates an error message that is logged to the system log file.

Parameters:

- string – text of the error message

nxa.SpinTelegramLog
This function forces to archive the current telegram log file and creates a new one.

Returns:

- bool – Indicates whether the operation was successful or not.

nxa.SpinSystemLog
This function forces to archive the current system log file and creates a new one.

Returns:

- bool – Indicates whether the operation was successful or not.

nxa.GetLastErrorText
This function retrieves the last error text as string.

Returns:

- string – The last error text.

nxa.GetLastErrorCode
This function retrieves the last error code as integer.

Returns:

- number – The last error code.

nxa.GetValue | nxa.Value
This function queries the current value of a Server Item. If the quality of the item is UNCERTAIN, it will return the

given default value. Note that this function reads the current value of the item that is stored within the server – it will not send a dedicated read request to the field device.

Parameters:

- string – itemID of the item that shall be queried
- variant – default value (optional)

Returns:

- variant – current value of the item

nxa.SetValue
This function sets the value of the specified Server Item. An optional delay can also be specified. Note that the new value will only be set within the server – it will not be forwarded to the field device.

Parameters:

- string – itemID of the item that shall be set
- variant – new value that shall be set
- number – delay in milliseconds (optional)

nxa.SetItemData
This function sets the value of the specified Server Item. Compared to the function “nxa.SetValue”, the time stamp and the source information can also be set. Note that the new value will only be set within the server – it will not be forwarded to the field device.

Parameters:

- string – itemID of the item that shall be set
- variant – new value that shall be set
- date – time stamp information (local time) for the value. This value is also stored in the historical database.
- string – source information (e.g. “SYS:LUA;SRC:MyDevice”)

nxa.ResetItemValue
This function restes the value of the specified Server Item i.e. its quality is set to “UNCERTAIN”.

Parameters:

- string – itemID of the item that shall be reset
- date – time stamp information (local time) for the value. This value is also stored in the historical database (optional).
- string – source information (e.g. “SYS:LUA;SRC:MyDevice”) (optional)

nxa.ReadValue
This function actively reads the value of the specified item. Compared to getting a value, a value read also sends a read request to the field device. An optional delay can also be specified. It is possible to delay this.

Parameters:

- string – itemID of the item that shall be read
- number – delay in milliseconds (optional)

Returns:

- variant – current value of the item

nxa.WriteValue
This function actively writes the given value of the specified item. Compared to setting a value, a value write will also propagate the value change to the field device. An optional delay can also be specified.

Parameters:

- string – itemID of the item that shall be written
- variant – new value that shall be written
- number – delay in milliseconds (optional)

nxa.ItemExists

This function checks whether the given Item is present in the OPC tree.

Parameters:

- string – Item ID of the Item that shall be tested.

Returns:

- bool – indicates whether the item exists or not.

nxa.IsValidValue
This function verifies whether the Item has a valid value (i.e. a quality of “GOOD”).

Parameters:

- string – Item ID of the Item that shall be tested.

Returns:

- bool – indicates whether the value is valid or not.

nxa.GetPropertyValue | nxa.PropertyValue | nxa.Property
This function reads the value of a property of a server item.

Parameters:

- string – itemID of the item that contains the property
- string – propertyID of the property that shall be read

Returns:

- variant – current value of the property

nxa.SetPropertyValue | nxa.SetProperty
This function writes a new value of a property of a server item. If the property is not writable, the function return false.

Parameters:

- string – itemID of the item that contains the property
- string – propertyID of the property that shall be written
- variant – new value of the property

Returns:

- bool – true when the operation was successful

nxa.ReadValues
Sends a KNX read requests to all KNX group addresses or to all KNX group addresses with activated “ReadOnReconnect” flag.

Parameters:

- bool – If true, the server sends a KNX read request to all KNX group addresses with activated “ReadOnReconnect” flag – otherwise a read request is sent to all KNX group addresses.

nxa.GetItemID | nxa.ItemID
This function is used to retrieve the itemID of the Server Item that is addressed by the given KNX address and the IP address of the KNX gateway.

Parameters:

- string – KNX address
- string – IP address of the KNX gateway (optional)

Returns:

- string – itemID of the item that is specified by the KNX address and the IP address of the KNX gateway

xdb.SetData
This function can be used to store any kind of persistent data. The data is also available after server restart.

Parameters:

- string – key of the value
- any – value that shall be stored in a persistent database

Returns:

- bool – true on success, false otherwise

xdb.ResetData
This function can be used to delete a key/value pair within the persistent database.

Parameters:

- string – key of the value

Returns:

- bool – true on success, false otherwise

xdb.GetData
This function can be used to retrieve data that is stored in the persistent database. The data is also available after server restart.

Parameters:

- string – key of the value that shall be retrieved

Returns:

- Any – value that is stored with the specified key

nxa.AddVar
This function adds a global variable within the LUA engine. The data is not persistent i.e. it is lost after server restart.

Parameters:

- string – name of the variable
- nxa.type – Data type of the variable (e.g. “nxa.String”)

nxa.GetVar
This function can be used to retrieve data of the given global variable. The data is not persistent i.e. it is lost after server restart.

Parameters:

- string – name of the variable

Returns:

- Any – value that is stored in the global variable the specified key

nxa.SetVar
This function can be used to set the value of the global variable. The data is not persistent i.e. it is lost after server restart.

Parameters:

- string – name of the variable
- any – value that shall be stored in the variable

Returns:

- bool – true on success, false otherwise

nxa.ClearVar
This function can be used to delete a key/value pair within the persistent database.

Parameters:

- string – name of the variable

Returns:

- bool – true on success, false otherwise

nxa.AddDirectLink
Adds a unidirectional link between the source and destination Item. This means that all requests (write, read, and set requests) are forwarded from the source to the destination item.

Parameters:

- string – source ItemID
- string – destination ItemID

nxa.RemoveDirectLink
Removes a unidirectional link between two items.

Parameters:

- string – source ItemID
- string – destination ItemID

nxa.AddDirectBiLink
Adds a bidirectional link between the two Items. This means that all requests (write, read, and set requests) are forwarded from one Item to the other.

Parameters:

- string – ItemID of Item 1
- string – ItemID of Item 2

nxa.RemoveDirectBiLink
Removes a bidirectional link between two items.

Parameters:

- string – ItemID of Item 1
- string – ItemID of Item 2

nxa.MakeItemShadowCopy
Creates a new item that behaves like a copy of the original one i.e. all requests (write, read, and set requests) to the original one are forwarded to the copy and vice versa.

Parameters:

- string – ItemID of the source Item
- string – The new of the new item that acts as copy
- string – The path of the hierarchy where the new item shall be located
- string – Deliminator that is used for specifying the path

nxa.MakeTreeShadowCopy

Parameters:

- SourceBranchPath
- DestinationBranchPath

4.6.2. NXA functions for task handling

This set of functions are used in combination with tasks. Some functions provide helper functionality that are only invokable within LUA functions that are executed by task (cf. Section 4.5.2). Furthermore, there are functions included that can be used to add and delete task definitions directly within LUA scripts. Using them provides the opportunity to dynamically create task that are not listed within the task definition file. Note that adding and deleting of task can only be done during server start up (e.g. within the callback “OnInitEvent”).

nxa.SourceItemID | nxa.SourceVar
This function can be used to query the ID of the source item that triggered the corresponding task. This function can only be used within LUA functions that are invoked by tasks.

Returns:

- string – itemID of the source item that triggered the task

nxa.DestinationItemID
This function can be used to query the ID of the destination item that is configured within the corresponding task. This function can only be used within LUA functions that are invoked by tasks.

Returns:

- string – itemID of the destination item that is configured within the task

nxa.SetDestinationValue

This function can be used to set a new value for the destination item that is configured within the corresponding task. This function can only be used within LUA functions that are invoked by tasks. Note that the new value will only be set within the server – it will not be forwarded to the field device.

Parameters:

- variant – new value of the destination item

nxa.WriteDestinationValue
This function can be used to write a new value to the destination item that is configured within the corresponding task. Compared to setting a value, a value write will also propagate the value change to the field device. This function can only be used within LUA functions that are invoked by tasks.

Parameters:

- variant – new value of the destination item

nxa.ReadDestinationValue
This function can be used to read the current value of the destination item that is configured within the corresponding task. This function can only be used within LUA functions that are invoked by tasks.

Returns:

- variant – current value of the destination item

nxa.InputValue | nxa.SourceValue
This function can be used to query the value of the source item that triggered the corresponding task. This function can only be used within LUA functions that are invoked by tasks.

Returns:

- variant – current value of the source item

nxa.ExecuteDelayedScript
This function executes a LUA script that is specified as String parameter. The execution can optionally be delayed.

Parameters:

- string – name of the LUA script
- number – delay in milliseconds (optional)

nxa.AddWriteTask
This function adds a new “WRITE” task definition during runtime.

Parameters:

- string – itemID of the source Server Item that triggers the task
- string – itemID of the destination Server Item that is changed by the task
- bool – corresponds to the “OnReceive” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSend” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSet” parameter of the task definition (cf. Section 4.5.2)
- number – time interval in milliseconds that the task will be delayed when triggered
- variant – if used, this value will be used instead of the value of the source item (optional)

nxa.AddReadTask
This function adds a new “READ” task definition during runtime.

Parameters:

- string – itemID of the source Server Item that triggers the task
- string – itemID of the Server Item that has to be read
- bool – corresponds to the “OnReceive” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSend” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSet” parameter of the task definition (cf. Section 4.5.2)
- number – time interval in milliseconds that the task will be delayed when triggered

nxa.AddSetTask

This function adds a new “SET” task definition during runtime.

Parameters:

- string – itemID of the source Server Item that triggers the task
- string – itemID of the destination Server Item that is changed by the task
- bool – corresponds to the “OnReceive” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSend” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSet” parameter of the task definition (cf. Section 4.5.2)
- number – time interval in milliseconds that the task will be delayed when triggered
- variant – if used, this value will be used instead of the value of the source item (optional)

nxa.AddScriptTask
This function adds a new “SCRIPT” task definition during runtime.

Parameters:

- string – sourceItemID
- string – destinationItemID
- bool – corresponds to the “OnReceive” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSend” parameter of the task definition (cf. Section 4.5.2)
- bool – corresponds to the “OnSet” parameter of the task definition (cf. Section 4.5.2)
- number – time interval in milliseconds that the task will be delayed when triggered
- string – LUA script name that will be executed when the task is triggered

nxa.AddVarTask
This function can be used to add a task that is triggered if a variable changes its value.

Parameters:

- string – name of the variable that shall be the trigger for the task.
- string – name of the LUA function that shall be invoked.
- number – an optional delay for executing the function.

Returns:

- bool – true on success, false otherwise

4.6.3. NXA functions for KNX In / Out Conversion

This set of functions are used in combination In / Out Converter definitions in KNX telegram definitions. Some functions provide helper functionality that are only invokable within such LUA conversion functions that are executed by such an item.

To still have the original value available, two more data points are added. They have extended Item IDs with “.IN” and “.OUT”.

The corresponding LUA function for IN conversion would look like that:

```
function InConv()
  Destination.SetValue(nxa.LowByte(Source.Value()))
end
```

! “Destination” would be the object where you would write the result of your conversion. For your InConv() the IN data point would be the receiving item. “SetValue()” is the function to call to push the value into the destination object without sending a telegram. “Source” in IN-convert-function is the object with the original value. “Value()” returns the actual value of the source object.

This means, in the above example the IN data point always will filter the lower byte off the original value.

In an IN-convert-function:

- Source - the original data item
- Destination - the IN data point

Here an example for an out converter:

```
function OutConv()
  Destination.WriteValue(nxa.LowByte(Source.Value()))
end
```

! “Destination” would be the object where you would write the result of your conversion. In an OUT-convert-function it would be the original data item. “WriteValue()” is the function to call to push the value into the destination object with sending a telegram. “Source” in Out-convert-function is the object with the original value. “Value()” returns the actual value of the source object.

The .OUT data point was set by a client, the LUA function OutConv() was called and the low byte value was set to the original data point. The difference between “SetValue()” and “WriteValue()” is, that in case of the latter one a telegram is generated.

In an OUT-convert-function:

- Source - the OUT data item
- Destination - the original data point

Destination.ID
Gets the ID of “Destination” object. e.g. Destination.ID() in an IN-convert-function would return the ID of the IN data point.

Returns:

- string – itemID of Destination object

Source.ID
Gets the ID of “Source” object. e.g. Source.ID() in an IN-convert-function would return the ID of the original data item.

Returns:

- string – itemID of Source object

Source.Value
Gets the actual value of “Source” object.

Returns:

- variant – actual value of Source object

Destination.Property
Gets the value of any property of “Destination” object. e.g. Destination.Property(0) in an IN-convert-function would return the canonical data type of the IN data point.

Parameters:

- number – propertyID

Returns:

- variant – value of proporty of Destination object

Source.Property
Gets the value of any property of “Source” object. e.g. Source.Property(0) in an IN-convert-function would return the canonical data type the original data item.

Parameters:

- number – propertyID

Returns:

- variant – value of proporty of Source object

Destination.ReadValue
Generates a read telegram in the “Destination” object.

Destination.SetValue
Sets the value of the “Destination” object, but does not generate any KNX telegram.

Parameters:

- variant – new value of the Destination object

Destination.WriteValue
Sets the value of the “Destination” object and generates a KNX telegram.

Parameters:

- variant – new value of the Destination object

4.6.4. NXA functions for date and time processing

nxa.Now
Return the current date and time.

Returns:

- Date – Current date and time

nxa.TimeOnly
Returns the current time (without data information) for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- Date – Current time (without the data information)

nxa.DateOnly
Returns the current date (without time information) for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- Date – Current date (without the time information)

nxa.Year
Returns the year for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Years

nxa.Month
Returns the month for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Months

nxa.Day
Returns the day for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Days

nxa.Hour
Returns the hour for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Hours

nxa.Minute
Returns the minute for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Minutes

nxa.Second
Returns the second for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Seconds

nxa.DayOfWeek
Returns the day of the week for the given date and time.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- number – Day of week

nxa.MakeDate
Creates a date variable.

Parameters:

- number – Year
- number – Month
- number – Day
- number – Hour
- number – Minute
- number – Second

Returns:

- Date – The created date variable

nxa.MakeTimeOnly
Creates a date variable that only contains time information.

Parameters:

- number – Hour
- number – Minute
- number – Second

Returns:

- Date – The created date variable

nxa.AddDate
Adds one date object to another.

Parameters:

- Date – First operand.
- Date – Second operand.

Returns:

- Date – the result of der opertation.

nxa.DiffDate

Subtracts one date object from another.

Parameters:

- Date – First operand.
- Date – Second operand.

Returns:

- Date – the result of der opertation.

nxa.DateToString
Converts a date object into its string representation.

Parameters:

- Date – The data/time variable that shall be converted.

Returns:

- String – String representation of the date

nxa.StringToDate
Converts a string into a date object.

Parameters:

- String – The string representation that should be converted to a date.

Returns:

- Date – The converted date

nxa.SetDate
Sets the given date.

Parameters:

- Date – The data/time variable that shall be set.

Returns:

- number – Day of week

nxa.SetTimeOnly
Sets the given time.

Parameters:

- Date – The time variable that shall be set.

Returns:

- number – Day of week

nxa.ExtTime
Returns the current time in the format "'HH:MM:SS.MMM"'.

Returns:

- String – String representation of current time

nxa.Tick
Returns the number of milliseconds that have elapsed since the system was started.

Returns:

- Number – number of milliseconds that have elapsed since the system was started

4.6.5. Overview about the functions of the nxa LUA module to extract data

nxa.LowByte
Extracts the low byte of the value.

Parameters:

- number – value

Returns:

- number – low byte of the value

nxa.HiByte
Extracts the high byte of the value.

Parameters:

- number – value

Returns:

- number – high byte of the value

nxa.LowWord
Extracts the low word of the value.

Parameters:

- number – value

Returns:

- number – low word of the value

nxa.HiWord
Extracts the high word of the value.

Parameters:

- number – value

Returns:

- number – high word of the value

nxa.IsBitSet
Checks if bit is set.

Parameters:

- number – value
- number – bit position

Returns:

- boolean – if bit was set it will be true, otherwise false

nxa.SetBit
Set bit at certain position.

Parameters:

- number – value
- number – bit position

Returns:

- number – result of changed bit

nxa.ResetBit
Reset bit at certain position.

Parameters:

- number – value
- number – bit position

Returns:

- number – result of changed bit

nxa.ResetBit
Reset bit at certain position.

Parameters:

- number – value
- number – bit position

Returns:

- number – result of changed bit

nxa.SetLoByte
Sets the low byte in the value.

Parameters:

- number – value
- number – byte value

Returns:

- number – result of changed value

nxa.SetHiByte
Sets the high byte in the value.

Parameters:

- number – value
- number – byte value

Returns:

- number – result of changed value

nxa.SetLoWord
Sets the low low word in the value.

Parameters:

- number – value
- number – word value

Returns:

- number – result of changed value

nxa.SetHiWord
Sets the high word in the value.

Parameters:

- number – value
- number – word value

Returns:

- number – result of changed value

4.6.6. Overview about the bit wise functions of the nxa LUA module to extract data

nxa.And
Bitwise AND of two numbers.

Parameters:

- number – value1
- number – value2

Returns:

- int – result of bit wise AND of value1 and value2

nxa.Or
Bitwise Or of two numbers.

Parameters:

- number – value1
- number – value2

Returns:

- int – result of bit wise Or value1 and value2

nxa.Xor
Bitwise Xor of two numbers.

Parameters:

- number – value1
- number – value2

Returns:

- int – result of bit wise Xor value1 and value2

nxa.Not
Bitwise Not of number.

Parameters:

- number – value

Returns:

- int – result of bit wise Not value1 and value2

4.6.7. NXA functions for creating virtual links and Server Items

These functions can be used to create user-defined Server Items (also referred to as Custom Items). These Custom Items can be organized in a user-defined hierarchy that is integrated into the Server Item Tree. Using custom items provides the most flexible way to add control functionality – they can be used the same way as real Server Items. The only difference is that they are only available in the server's virtual data model – there is no physical datapoint that is connected to it. Therefore, Custom Items can be seen as virtual datapoints.

Note that Custom Items can only be create and deleted during server start up (e.g. within the "OnInitEvent" callback).

nxa.AddCustomItem
This function adds a new Custom Item during the initialization phase of the server and returns its ID. In general, this function shall be used within the "OnInitEvent" callback only.

Parameters:

- string – name of the Custom Item
- string – description of the Custom Item
- nxa.access – access rights of the Custom Item
- nxa.type – the data type of the Custom Item
- string – delimiter that is used to build the item ID (optional)
- string – path1 (optional)
- string – path2 (optional)
- ...
- string – pathN (optional)

Returns:

- string – itemID of the new Custom Item

Example:

```
nxa.AddCustomItem(
  "TestItem",
  "my test item description",
  nxa.access.All,
  nxa.type.Real,
  "/",
  "MySubDir",
  "MySubDir2")
```

This will create a Custom Item called "TestItem" within the Server Item Tree under the sub tree "Custom/MySubDir/MySubDir2". Thus, the itemID of the new Custom Item is "Custom/MySubDir/MySubdir2/TestItem".

nxa.AddExtCustomItem
This function adds a new Custom Item during the initialization phase of the server and returns its ID. In general, this function shall be used within the “OnInitEvent” callback only. Compared to the function “nxa.AddCustomItem”, the flags for “Persistent”, “Historical”, and “Synchronziation” can also be set.

Parameters:

- string – name of the Custom Item
- string – description of the Custom Item
- nxa.access – access rights of the Custom Item
- nxa.type – the data type of the Custom Item
- bool – sets the persistent flag of the Custom Item
- bool – sets the historical flag of the Custom Item
- bool – sets the synchronization flag of the Custom Item
- string – delimiter that is used to build the item ID (optional)
- string – path1 (optional)
- string – path2 (optional)
- . . .
- string – pathN (optional)

Returns:

- string – itemID of the new Custom Item

nxa.AddSysCustomItem
This function adds a new System Custom Item during the initialization phase of the server and returns its ID. Compared to a normal custom item, a system custom item can be placed anywhere in the item tree. In general, this function shall be used within the “OnInitEvent” callback only.

Parameters:

- string – name of the custom item
- string – description of the custom item
- nxa.access – access rights of the custom item
- nxa.type – the data type of the custom item
- string – engineering units of the custom item (optional)
- number – minimum value that is allowed for the custom item (optional)
- number – maximum value that is allowed for the custom item (optional)
- string – delimiter that is used to build the item ID (optional)
- string – path1 (optional)
- string – path2 (optional)
- . . .
- string – pathN (optional)

Returns:

- string – itemID of the new Custom Item

nxa.AddExtSysCustomItem
This function adds a new System Custom Item during the initialization phase of the server and returns its ID. Compared to a normal custom item, a system custom item can be placed anywhere in the item tree. In general, this function shall be used within the “OnInitEvent” callback only. Compared to the function “nxa.AddSysCustomItem”, the flags for “Persistent”, “Historical”, and “Synchronziation” can also be set.

Parameters:

- string – name of the custom item
- string – description of the custom item
- nxa.access – access rights of the custom item

- nxa.type – the data type of the custom item
- bool – sets the persistent flag (optional)
- bool – sets the historical flag (optional)
- bool – sets the synchronize flag (optional)
- string – engineering units of the custom item (optional)
- number – minimum value that is allowed for the custom item (optional)
- number – maximum value that is allowed for the custom item (optional)
- string – delimiter that is used to build the item ID (optional)
- string – path1 (optional)
- string – path2 (optional)
- . . .
- string – pathN (optional)

Returns:

- string – itemID of the new Custom Item

nxa.AddItemLink
This function links a source item to a destination item. This means that the value of the source item will be forwarded to the destination item.

Parameters:

- string – itemID of the source item
- string – itemID of the destination item
- number – delay in milliseconds

nxa.AddDirectLink
This function directly links a source item to a destination item. This means that the value of the source item will be forwarded to the destination item. Compared to "'nxa.AddItemLink"', loops are detected.

Parameters:

- string – itemID of the source item
- string – itemID of the destination item

nxa.RemoveDirectLink
This function removes a direct link again.

Parameters:

- string – itemID of the source item
- string – itemID of the destination item

nxa.AddDirectBiLink
This function bi-directly links a source item to a destination item. This means that the value of the source item will be forwarded to the destination item and vice versa. Compared to "'nxa.AddItemLink"', loops are detected.

Parameters:

- string – itemID of the source item
- string – itemID of the destination item

nxa.RemoveDirectBiLink
This function removes a bi-directional link again.

Parameters:

- string – itemID of the source item
- string – itemID of the destination item

nxa.AddItemEvent
This function adds a new event that is triggered whenever the given item changes.

Parameters:

- string – itemID of the source Server Item that triggers the event
- bool – similar to the “OnReceive” parameter of the task definition (cf. Section 4.5.2), this parameter indicates that the event is triggered if a read request is received on the given item
- bool – similar to the “OnSent” parameter of the task definition (cf. Section 4.5.2), this parameter indicates that the event is triggered if a new value is sent to the given item
- bool – similar to the “OnSet” parameter of the task definition (cf. Section 4.5.2), this parameter indicates that the event is triggered if a new value is set
- number – delay in milliseconds
- string – script that is executed when the event is triggered

nxa.AddItemTemplate
This function creates a new template for Custom Items and adds it to the template list inside the server.

Returns:

- number – ID that uniquely identifies the template

nxa.AddPropertyTemplate
This function can be used to add a property to an item template. The template has to be created with “nxa.AddItem Template” before.

Parameters:

- number – ID that identifies the template to which the property has to be added
- number – ID that uniquely identifies the property within the template. The number room for the ID is 7000-7999.
- string – name of the property
- nxa.type – the data type of the property
- variant – default value of the property

nxa.AttachTemplate
This function assigns a template to a Custom Item.

Parameters:

- string – itemID of the item to which the template has to be attached
- number – ID that identifies the template that has to be attached to the specified item

nxa.DetachTemplate
This function detaches a template again.

Parameters:

- string – itemID of the item to which the template has to be attached

4.6.8. NXA LUA Apps

All LUA functions that are invoked by server tasks as well as all predefined event callbacks (cf. Section 4.6.10) are called in the same thread context. This means that these functions are not executed independent of each other – they are executed one by one. Especially if a lot of functionality is implemented in a LUA function i.e. if it contains a lot of LUA code, this may be a disadvantage because a long execution of a LUA function may block other LUA function.

In order to be able to implement time consuming tasks, so called LUA Apps are available. LUA Apps are executed within a separate thread within the LUA engine. This means that LUA Apps are executed to some extend in parallel without block each other.

LUA Apps are invoked using the following LUA function:

nxa.StartApp
Start a LUA App in a separate thread context.

Parameters:

- LUA file – name of the LUA file where the function is defined

- LUA function – LUA worker function that is invoked within a separate thread

After calling the “nxa.StartApp”, the given function is executed as App. If the function is returns, the App is automatically terminated. This means if a LUA App shall be able to do some periodic task, an event loop has to be included.

Since LUA Apps are invoke to some extent in parallel, the implemented functionality and the used resource must be thread-safe i.e. critical operations must be protected. Therefore, only the following functions can be used within LUA Apps:

- **nxa.LogInfo**
- **nxa.LogWarning**
- **nxa.LogError**
- **nxa.IsInitialized**
- **nxa.IsRunning**
- **nxa.IsSimulation**
- **nxa.IsActiveServer**
- **nxa.IsMainServer**
- **nxa.IsBackupServer**
- **nxa.GetValue** | **nxa.Value**
- **nxa.SetValue**
- **nxa.SetItemData**
- **nxa.WriteValue**
- **nxa.ReadValue**
- **nxa.IsValidValue**
- **nxa.WorkspaceName**
- **nxa.RootPath**
- **nxa.WorkspacePath**
- **nxa.ScriptFilesPath**
- **nxa.DataFilesPath**
- **nxa.LogFilesPath**
- **nxa.ProjectFilesPath**
- **nxa.EventFilesPath**
- **nxa.ConfigFilesPath**
- **nxa.GetItemID**
- **nxa.Sleep**
- **nxa.Now**
- **nxa.TimeOnly**
- **nxa.DateOnly**
- **nxa.Year**
- **nxa.Month**
- **nxa.Day**
- **nxa.Hour**
- **nxa.Minute**
- **nxa.Second**
- **nxa.DayOfWeek**
- **nxa.MakeDate**
- **nxa.MakeTimeOnly**
- **nxa.AddDate**
- **nxa.DiffDate**
- **nxa.DateToString**
- **nxa.SetDate**
- **nxa.SetTimeOnly**
- **nxa.GetVar**
- **nxa.SetVar**
- **nxa.ClearVar**
- **nxa.AddVarTask**
- **nxa.SourceVar**

- **nxa.GetPropertyValue | nxa.PropertyValue | nxa.Property**
- **nxa.SetPropertyValue | nxa.SetProperty**
- **nxa.GetLastErrorText**
- **nxa.GetLastErrorCode**
- **nxa.LowByte**
- **nxa.HiByte**
- **nxa.LowWord**
- **nxa.HiWord**
- **nxa.IsBitSet**
- **nxa.SetBit**
- **nxa.ResetBit**
- **nxa.SetLowByte**
- **nxa.SetHiByte**
- **nxa.SetLowWord**
- **nxa.SetHiWord**
- **nxa.And**
- **nxa.Or**
- **nxa.Xor**
- **nxa.Tor**
- **nxa.Not**

4.6.9. NXA functions for network communication

The LUA engine of the NETx BMS Server provides different runtime functions for network communication, e.g. e-mail, TCP, or FTP. In order to correctly send e-mails, the outgoing SMTP server has to be configured within the System Configuration file (cf. Section 4.2.2.5).

xcon.Close
Closes a connection.

Parameters:

- string – Handle that identifies the connection within the LUA script.

xcon.Create
This function can be used to create a socket in LUA.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- number – Socket type. Allowed values are nxa.xcon.UDP = 1, nxa.xcon.TCPClient = 2, nxa.xcon.HTTP = 4, nxa.xcon.RSS = 5, nxa.xcon.COM = 6, nxa.FTPClient = 8 TODO XXXX
- string – the IP address of the local interface. If empty, the IP is detected by the system.
- number – the local port that is used for the connection. If zero, the system selects one.
- string – the IP address or host name of the remote server.
- number – the remote port that is used to connect to.

xcon.CreateCOM
This function can be used to create a serial connection in LUA.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – The name of the COM port in the system (e.g. “COM1”).
- number – The baud rate of the serial port.
- number – The amount of data bits of the serial port.
- number – The used parity of the serial port.
- number – The amount of stop bits of the serial port.
- number – The used handshake protocol of the serial port.

- number – The used event char of the serial port.

xcon.CreateFTP
This function creates an FTP client socket.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – host name
- number – port number
- string – user name
- string – password

Returns:

- bool – initiation status; true if successful, otherwise NIL.

xcon.CreateHTTP
This function can be used to create an HTTP connection in LUA.

Parameters:

- string – Handle that identifies the connection within the LUA script.

xcon.CreateTCP
This function can be used to create a TCP socket in LUA.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – the IP address of the local interface. If empty, the IP is detected by the system.
- number – the local port that is used for the connection. If zero, the system selects one.
- string – the IP address or host name of the remote server.
- number – the remote port that is used to connect to.

xcon.CreateRSS
This function can be used to create an RSS feed in LUA.

Parameters:

- string – Handle that identifies the RSS feed within the LUA script.

xcon.CreateUDP
This function can be used to create a UDP socket in LUA.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – the IP address of the local interface. If empty, the IP is detected by the system.
- number – the local port that is used for the connection. If zero, the system selects one.
- string – the IP address or host name of the remote server.
- number – the remote port that is used to connect to.

xcon.Download
This function initiates an asynchronous FTP download. If a download is already taking place on this connection (identified by host, port, user, and password), the transfer is enqueued and commenced once the current transfer is finished. Provide `_OnDoneEvent` and `_OnErrorEvent` callback functions to be notified on transfer completion and errors (ref. section 4.6.10).

Parameters:

- string – connection handle
- string – remote file name
- number – local file name

Returns:

- bool – initiation status; true if successful, otherwise NIL.

xcon.IsConnected
This function determines whether the connection is established.

Parameters:

- string – Handle of the connection.

Returns:

- bool – Connection state.

xcon.Send
This function can be used to send data via an established connection.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – Data to be sent
- number – amount of characters to be send

Returns:

- bool – true when server is initialized, otherwise false.

xcon.SendAdminEmail
This function can be used to directly send an e-mail to the specified admin e-mail address.

Parameters:

- string – e-mail subject
- string – e-mail body

xcon.SendEmailTo
This function can be used to directly send an e-mail within LUA.

Parameters:

- string – e-mail address of the receiver
- string – e-mail subject
- string – e-mail body

xcon.SendSystemEmail
This function can be used to directly send an e-mail to the specified systen e-mail address.

Parameters:

- string – e-mail subject
- string – e-mail body

xcon.SendText
This function can be used to send data via an established connection.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – Data to be sent

Returns:

- bool – true when server is initialized, otherwise false.

xcon.SendTextTo
This function can be used to send data via an established connection.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – Data to be sent
- string – the IP address or host name of the remote server.
- number – the remote port that is used to connect to.

Returns:

- bool – true when server is initialized, otherwise false.

xcon.SendTo
This function can be used to send data via an established connection.

Parameters:

- string – Handle that identifies the connection within the LUA script.
- string – Data to be sent
- number – amount of characters to be send
- string – the IP address or host name of the remote server.
- number – the remote port that is used to connect to.

Returns:

- bool – true when server is initialized, otherwise false.

xcon.SendUserEmail
This function can be used to directly send an e-mail to the specified user e-mail address.

Parameters:

- string – e-mail subject
- string – e-mail body

xcon.Upload
This function initiates an asynchronous FTP upload. If an upload is already taking place on this connection (identified by host, port, user, and password), the transfer is enqueued and commenced once the current transfer is finished. Provide `_OnDoneEvent` and `_OnErrorEvent` callback functions to be notified on transfer completion and errors (ref. section 4.6.10).

Parameters:

- string – connection handle
- string – local file name
- number – remote file name

Returns:

- bool – initiation status; true if successful, otherwise NIL.

4.6.10. NXA event callbacks

In addition to several LUA functions, the NXA LUA library also provides several event callbacks that are invoked by the server at specific points in time. Within these callbacks, the user can integrate LUA scripts that are execute whenever the server invokes the corresponding callback. Using this way, control functionality can be implemented that shall be available for dedicated points in time.

OnInitEvent
This callback is invoked the during initialization of the server. Within this callback, initialization task (e.g. creating Custom Items) can be performed.

OnStartEvent
This callback is invoked during server start up. Since “OnStartEvent” is called after “OnInitEvent”, all items are already available and can be accessed through LUA scripts.

OnStopEvent
This callback is invoked during the shutdown process of the server.

OnSecondTimerEvent
This callback is invoked every second. It can be used to implement functionality that has to be triggered every second.

OnMinuteTimerEvent
This callback is invoked every minute. It can be used to implement functionality that has to be triggered every minute.

OnHourTimerEvent

This callback is invoked every hour. It can be used to implement functionality that has to be triggered every hour.

OnKNXGatewayConnectedEvent
This callback is invoked every time a KNX gateway has been connected. The parameter can be used to determine the connected gateway.

Parameters:

- string – IP address of the KNX gateway

OnKNXGatewayDisconnectedEvent
This callback is invoked every time a KNX gateway has been disconnected. The parameter can be used to determine the disconnected gateway.

Parameters:

- string – IP address of the KNX gateway

nxa.OnClientConnectedEvent
This function is invoked when a client is connected to the NETx BMS Server.

Parameters:

- clientType - client type (WEB, VNET, or OPC)
- clientName - client name
- source - client source (e.g. Voyager.5.0, OPC Client name or IP address of BMS Client)
- IPAddress - ip address of client
- user - user that is currently online

nxa.OnClientDisconnectedEvent
This function is invoked when a client is disconnected from the NETx BMS Server.

Parameters:

- clientType - client type (WEB, VNET, or OPC)
- clientName - client name
- source - client source (e.g. Voyager.5.0, OPC Client name or IP address of BMS Client)
- IPAddress - ip address of client
- user - user that is currently online

<connection_handle>_OnDoneEvent
This function is invoked when an FTP connection completed a transfer. Create a function for each connection to monitor and name it accordingly, e.g. for connection `ftp1` create the function `ftp1_OnDoneEvent`.

Parameters:

- string - source file name
- string - destination file name

<connection_handle>_OnErrorEvent
This function is invoked when an error occurs on an FTP connection. Create a function for each connection to monitor and name it accordingly, e.g. for connection `ftp1` create the function `ftp1_OnErrorEvent`.

Parameters:

- string - error message

4.6.11. NXA constants

Here, different constants are defined that are required as parameters for different NXA LUA functions.

NXA data types
These constants define the different data types that are available for Server Items.

- nxa.type.Integer
- nxa.type.Real
- nxa.type.Date

- nxa.type.String
- nxa.type.Boolean

NXA access rights
These constants define the different access rights that are available for Server Items.

- nxa.access.Readable
- nxa.access.Writeable
- nxa.access.All

4.6.12. Additional information to LUA implementation

The integrated LUA script engine also provides the opportunity to split the LUA implementation into separate source files. Using this way, it is possible to implement user-defined LUA functions that can be reused in other NETx BMS Server projects. In addition it keeps the LUA source code short, clearly arranged, and easy to maintain.

User-defined LUA script files have to be stored within the script directory of the workspace directory. To reuse a LUA script file, it has to be imported within the standard LUA file “nxaDefinitions.lua” by using the key work “require”.

Example:

```
require "MyLuaScriptFile"
```

This code line includes the file “MyLuaScriptFile.lua” into the default LUA script file – thus all functions that are defined there are available within the whole script engine. Note that is has to be ensured that a function name is used only once.

4.7. XCON Interfaces

XCON provides the system integrator a manifolded interface to different communication platforms. If it is sending simple emails or a deeply integrated communication via TCP/IP is just a matter of choice and not of possibility.

4.7.1. COM Interface

It is used to communicate with serial interface. It consists of five communication entries and a configuration branch. It needs to be enabled before it can be used.

Item Name:	**OUT**
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.OUT
Description:	The byte buffer to send to the COM interface shall be written to this item.

Item Name:	**IN**
Data Type:	STRING
Default value:	None
Access Rights:	Read only
Standard Path:	NETx.XCON.<COM#>.IN
Description:	The byte buffer returning from the the COM interface will be written to this item.

Item Name:	**Enabled**
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.Enabled
Description:	This item turns the COM interface on and off. It is responsible for creating a connection or dropping it.

Item Name:	Connected
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read only
Standard Path:	NETx.XCON.<COM#>.Connected
Description:	It shows whether the COM connection is opened or closed.

Item Name:	LastError
Data Type:	STRING
Default value:	empty string
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.LastError
Description:	If an error occurred, it will be written to this item.

4.7.1.1. Config

Item Name:	Device
Data Type:	STRING
Default value:	“COM1”
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.Config.Device
Description:	The item determines which COM interface shall be addressed when opening the connection.

Item Name:	Baudrate
Data Type:	INT4
Default value:	9600
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.Config.Baudrate
Description:	This item contains the communication speed of the serial interface. Standard baud rates are the following:

Table 4.1.: Standard Baud Rates

110	300	600
1200	2400	4800
9600	14400	19200
28800	38400	56000
57600	115200	

Item Name:	DataBits
Data Type:	INT4
Default value:	8
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<COM#>.Config.DataBits
Description:	This determines the number of data bits in each character. It can be 5 (for Baudot code), 7 (for true ASCII), or 8 (for any kind of data, as this matches the size of a byte). 8 data bits are almost universally used in newer applications. Values like 6 or 9 are used scarcely.

Item Name:	**Parity**
Data Type:	INT4
Default value:	0 (False)
Access Rights:	Read only
Standard Path:	`NETx.XCON.<COM#>.Config.Parity`
Description:	If set to another value but “0” an additional bit will be added to each character and a parity check will be performed during communication. These are the supported parity values:

Table 4.2.: Parity Values

Value	Meaning
0	No parity
1	Odd parity
2	Even parity
3	Mark parity
4	Space parity

Item Name:	**StopBits**
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	`NETx.XCON.<COM#>.Config.StopBits`
Description:	This determines the number of stop bits at the end of each character. It is used to detect the end of a character by the receiving device and to resynchronize. Slow electromechanical devices can demand up to 2 stop bits. These are the supported stop bit values:

Table 4.3.: Stop Bits

Value	Meaning
0	1.0 stop bits
1	1.5 stop bits
2	2.0 stop bits

Item Name:	**Handshake**
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	`NETx.XCON.<COM#>.Config.Handshake`
Description:	Set this if flow control via handshake is needed otherwise keep it at 0.

Item Name:	**EventChar**
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	`NETx.XCON.<COM#>.Config.EventChar`
Description:	This determines the character used to signal an end of a line (e.g. CHR(10) for Linefeed).

4.7.2. UDP Interface

Just like the COM interface it is used to communicate with the UDP interface. It consists of five communication entries and a configuration branch. It needs to be enabled before it can be used.

Item Name:	OUT
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.OUT
Description:	The byte buffer to send to the UDP interface shall be written to this item.

Item Name:	IN
Data Type:	STRING
Default value:	None
Access Rights:	Read only
Standard Path:	NETx.XCON.<UDP>.IN
Description:	The byte buffer read by the the UDP interface will be written to this item.

Item Name:	Enabled
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.Enabled
Description:	This item turns the UDP interface on and off. It is responsible for creating a connection or dropping it.

Item Name:	Connected
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read only
Standard Path:	NETx.XCON.<UDP>.Connected
Description:	It shows whether the UDP connection is opened or closed.

Item Name:	LastError
Data Type:	STRING
Default value:	empty string
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.LastError
Description:	If an error occurred, it will be written to this item.

4.7.2.1. Config

Item Name:	LocalHost
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.Config.LocalHost
Description:	This item shall contain the local IP address.

Item Name:	LocalPort
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.Config.LocalPort
Description:	This item contains the local port the UDP communication is supposed to work on.

Item Name:	RemoteHost
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.Config.RemoteHost
Description:	This item shall contain the IP address of the remote device to communicate with.

Item Name:	RemotePort
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<UDP>.Config.RemotePort
Description:	This item contains the remote port the UDP communication is supposed to work on.

4.7.3. TCP Interface

Just like the COM interface it is used to communicate with the TCP interface. It consists of five communication entries and a configuration branch. It needs to be enabled before it can be used.

Item Name:	OUT
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.OUT
Description:	The byte buffer to send to the TCP interface shall be written to this item.

Item Name:	IN
Data Type:	STRING
Default value:	None
Access Rights:	Read only
Standard Path:	NETx.XCON.<TCP>.IN
Description:	The byte buffer read by the the TCP interface will be written to this item.

Item Name:	Enabled
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.Enabled
Description:	This item turns the TCP interface on and off. It is responsible for creating a connection or dropping it.

Item Name:	Connected
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read only
Standard Path:	NETx.XCON.<TCP>.Connected
Description:	It shows whether the TCP connection is opened or closed.

Item Name:	LastError
Data Type:	STRING
Default value:	empty string
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.LastError
Description:	If an error occurred, it will be written to this item.

4.7.3.1. Config

Item Name:	LocalHost
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.Config.LocalHost
Description:	This item shall contain the local IP address.

Item Name:	LocalPort
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.Config.LocalPort
Description:	This item contains the local port the TCP communication is supposed to work on.

Item Name:	RemoteHost
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.Config.RemoteHost
Description:	This item shall contain the IP address of the remote device to communicate with.

Item Name:	RemotePort
Data Type:	INT4
Default value:	0
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<TCP>.Config.RemotePort
Description:	This item contains the remote port the TCP communication is supposed to work on.

4.7.4. HTTP Interface

The HTTP interface can be used to get information from a remote host into the NETx BMS Server. A simple HTTP request is sent to the remote host and its result is stored in another item. It can be read, analyzed, and more information taken off of it. It needs to be enabled before it can be used.

Item Name:	GET
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<HTTP>.GET
Description:	As soon as a HTTP Get command is set to this item it will be sent. Example: localhost/index.html Note that the prefix “http://” is missing and that the command includes the full path of the resource called.

Item Name:	PUT
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<HTTP>.PUT
Description:	Place any HTTP Put command here to send it immediately to the alien server. Example: localhost/index.html Note that the prefix “http://” is missing and that the command includes the full path of the resource called.

Item Name:	RESULT
Data Type:	STRING
Default value:	None
Access Rights:	Read only
Standard Path:	NETx.XCON.<HTTP>.RESULT
Description:	The result of the last sent HTTP request will be stored in this item. Example: <HTML><HEAD>My Page</HEAD><BODY>Hello World!</BODY ></HTML>

Item Name:	Enabled
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<HTTP>.Enabled
Description:	If this Server Item is set to “True” any Get or Put command will immediately be sent off to the alien HTTP server.

Item Name:	LastError
Data Type:	STRING
Default value:	empty string
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<HTTP>.LastError
Description:	If an error occurred during transmission it will be stored in here. The Server Item will keep this value. It will change on the next error or can be overwritten by script or any other means.

4.7.5. RSS Interface

The RSS interface allows the user to get information from an RSS provider. It needs to be enabled before it can be used. Writing an RSS URL will initialize the communication.

Item Name:	GET
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<RSS>.GET
Description:	The URL for the RSS feed shall be placed here. Example: http://www.netxautomation.biz/netx/news_en.xml

Item Name:	CHANNEL
Data Type:	INT4
Default value:	-1
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<RSS>.CHANNEL
Description:	The channel is the zero based index of the channel one would like to get out of the RSS document. “-1” results into all channels being returned.

Item Name:	RESULT
Data Type:	STRING
Default value:	0 (False)
Access Rights:	Read only
Standard Path:	NETx.XCON.<RSS>.RESULT
Description:	The result of the RSS request will be stored in this item.

Item Name:	Enabled
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<RSS>.Enabled
Description:	If set to “True” any Get command will immediately be sent off to the alien RSS server.

Item Name:	LastError
Data Type:	STRING
Default value:	empty string
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<RSS>.LastError
Description:	If an error occurred, it will be written to this item.

4.7.6. EMAIL Interface

The EMAIL interface can send email to either a predefined address group or to any receiver placed in the “TO” item. Connection definitions concerning the SMTP server have to be made previously (ref. System Settings 4.2.2.5). Depending on which “SendTo” item is set to true the email interface will send the content to different addresses.

Item Name:	**SUBJECT**
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.SUBJECT
Description:	Place the subject of the email here.

Item Name:	**BODY**
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.BODY
Description:	This item contains the content of the email.

Item Name:	**TO**
Data Type:	STRING
Default value:	None
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.TO
Description:	Add email addresses here for sending the message to them with the “SendTo” trigger.

Item Name:	**SendTo**
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.SendTo
Description:	Once this item is set to “True” the server will send the email (composed in “SUBJECT” and “BODY”) to the address set in “TO”.

Item Name:	**SendToAdmin**
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.SendToAdmin
Description:	Once this item is set to “True” the server will send the email (composed in “SUBJECT” and “BODY”) to the address set in the system configuration in parameter EMAIL.AdminMailingList.

Item Name:	**SendToSystem**
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.SendToSystem
Description:	Once this item is set to “True” the server will send the email (composed in “SUBJECT” and “BODY”) to the address set in the system configuration in parameter EMAIL.SystemMailingList.

Item Name:	**SendToUser**
Data Type:	BOOL
Default value:	0 (False)
Access Rights:	Read and Write
Standard Path:	NETx.XCON.<EMAIL>.SendToUser
Description:	Once this item is set to “True” the server will send the email (composed in “SUBJECT” and “BODY”) to the address set in the system configuration in parameter EMAIL.UserMailingList.

4.8. Server logging

For maintenance, analysing, and auditing purposes, logging of the system's activities is one of utmost importance. Therefore, the NETx BMS Server provides different logging facilities that are described in the remainder of this section.

4.8.1. System logging

Location:

```
<WorkspaceDirectory>\LogFiles\nxaOPCSystem.35.log
```

Within this log file, all relevant event and system messages that are logged by the server application are stored. During server start up, a new, empty system log file is created by the server. However, the old system log files are not lost – after each server start up, the old system log file is renamed to "nxaOPCSystem.35.<timestamp>.log" where <timestamp> is the time of the current server start up.

If an unexpected server behaviour or other problems during runtime are observed, this server log file is a good starting point for analysing the reason of the problem. Since almost all information of the server is logged within this file, analyzing this file helps to solve the issue in most cases.

Example:

```
INFO;05/03/12 08:11:35.234;SERVER_ENGINE;Licensed Gateway Number: 32;0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'OPC.GroupAddressType' = '2Level';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'OPC.AsyncReadFromDevice' = 'FALSE';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'OPC.AsyncWriteConfirmation' = 'FALSE';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'OPC.AsyncRefreshFromDevice' = 'FALSE';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'OPC.AsyncTimeout' = '10';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'KNX.Timeout' = '3';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'SYS.EnableDeviceManager' = 'TRUE';0
INFO;05/03/12 08:11:35.249;SERVER_ENGINE;Option 'SYS.GatewayConnectionTimeout' = '100';0
```

Each line corresponds to one logging entry.

Column 1 – LogLevel
This column defines the log level of the event. This can be "INFO", "WARNING", or "ERROR".

Column 2 – Timestamp
This column specifies the point in time when the event occurred.

Column 3 – ModuleName
This column defines the module name that causes the event.

Column 3 – Message
This column shows the text of the event message.

4.8.2. Telegram logging

Location:

```
<WorkspaceDirectory>\LogFiles\nxaOPCTelegram.35.log
```

The NETx BMS Server provides the opportunity to log all incoming and outgoing KNX network telegrams that are received or sent by the server. If activated, the content of the telegrams as well as additional information (e.g. the time of arrival) are logged to the file mentioned above. Obviously, storing all telegrams may need a high amount of memory space. Therefore, this logging information is stored in binary format instead of storing it in clear. To view the content of the telegram log file, the telegram history explorer (cf. Section 3.4.3) can be used. Furthermore, the parameter "SYS.MaxSizeOfTelegramLogFile" can be used to define the maximum file size of the telegram log file (cf. Section 4.2.2) – if the this file size is exceeded, the oldest entries are overwritten.

Similar to the system log file, an empty telegram log file is created at system start up – the old telegram log file is renamed to "nxaOPCTelegram.35.<timestamp>.log" where <timestamp> is the time of the current server start up.

! In order to enable telegram logging, the system parameter "SYS.UseTelegramDataFile" has to be set to "ON".

4.9. Supported data types

Every row of the below-mentioned table (except the column “description”) defines a data type.

4.9.1. Server data types

Table 4.4.: Server Data Types

Server Data Type	Size	(U)nsigned (S)igned
BOOL	1 Bit	
STR	Variable	
BYTE	1 Byte	Unsigned
CHAR	1 Byte	Signed
INT8	1 Byte	Signed
UINT8	1 Byte	Unsigned
INT16	2 Byte	Signed
UINT16	2 Byte	Unsigned
INT32	4 Byte	Signed
UINT32	4 Byte	Unsigned
INT64	8 Byte	Signed
UINT64	8 Byte	Unsigned
FLOAT	4 Byte	
DOUBLE	8 Byte	
DATE		
TIME		

4.9.2. KNX data types

The NETx BMS Server supports all KNX datapoint types (DPT) as defined in the KNX Standard 2.1 in Section 3.7.2. In addition, the EIS types are also supported.

Table 4.5.: KNX EIS data types

Data size	EIS Type	(S)igned / (U)nsigned	Unit	Meaning
1BIT	EIS1			Switch
4BIT	EIS2			Dim object Bit 0=Up/Down Bit 1..3=Speed if Bit1..4=0 Stop
3BYTE	EIS3			Time
3BYTE	EIS4			Date(Attention: century 2-digits)
2BYTE	EIS5		°C	Temperature
2BYTE	EIS5		K	Temperature difference
2BYTE	EIS5		K/h	Temperature gradient
2BYTE	EIS5		Lux	Light intensity
2BYTE	EIS5		m/s	Wind velocity
2BYTE	EIS5		Pa	Air pressure
2BYTE	EIS5		s	Time difference
2BYTE	EIS5		ms	Time difference
2BYTE	EIS5		mV	Voltage
2BYTE	EIS5		mA	Current
1BYTE	EIS6		%	Relative Luminance (0...100)
1BYTE	EIS6		%	Relative Humidity (0...100)
1BYTE	EIS6		°	Wind direction (0...360)
1BYTE	EIS6	U		8-Bit unsigned (unofficial Type)
1BYTE	EIS6	S		8-Bit signed (unofficial Type)
1BIT	EIS7			Motor Motion
2BIT	EIS8			Priority Control
4BYTE	EIS9		m/s_2	Acceleration
4BYTE	EIS9		rad/s_2	
4BYTE	EIS9		J/mol	
4BYTE	EIS9		1/s	
4BYTE	EIS9		mol	

Continued on next page . . .

Table 4.5.: KNX EIS data types

Data size	EIS Type	(S)igned / (U)nsigned	Unit	Meaning
4BYTE	EIS9			
4BYTE	EIS9		rad	
4BYTE	EIS9		°	
4BYTE	EIS9		Js	
4BYTE	EIS9		rad/s	
4BYTE	EIS9		m_2	
4BYTE	EIS9		F	
4BYTE	EIS9		C/m_2	
4BYTE	EIS9		C/m_3	
4BYTE	EIS9		m_2/N	
4BYTE	EIS9		S	
4BYTE	EIS9		S/m	
4BYTE	EIS9		kg/m	
4BYTE	EIS9		C	
4BYTE	EIS9		A	
4BYTE	EIS9		A/m_2	
4BYTE	EIS9		Cm	
4BYTE	EIS9		C/m_2	
4BYTE	EIS9		V/m	
4BYTE	EIS9		C	
4BYTE	EIS9		C/m_2	
4BYTE	EIS9		C/m_2	
4BYTE	EIS9		V	
4BYTE	EIS9		V	
4BYTE	EIS9		Am	
4BYTE	EIS9		V	
4BYTE	EIS9		J	
4BYTE	EIS9		N	
4BYTE	EIS9		1/s	
4BYTE	EIS9		rad/s	
4BYTE	EIS9		J/K	
4BYTE	EIS9		W	
4BYTE	EIS9		J	
4BYTE	EIS9		Ohm	
4BYTE	EIS9		m	
4BYTE	EIS9		J	
4BYTE	EIS9		cd/m_2	
4BYTE	EIS9		lm	
4BYTE	EIS9		cd	
4BYTE	EIS9		A/m	
4BYTE	EIS9		Wb	
4BYTE	EIS9		T	
4BYTE	EIS9		Am	
4BYTE	EIS9		T	
4BYTE	EIS9		A/m	
4BYTE	EIS9		A	
4BYTE	EIS9		kg	
4BYTE	EIS9		kg/s	
4BYTE	EIS9		N/s	
4BYTE	EIS9		rad	
4BYTE	EIS9		°	
4BYTE	EIS9		W	
4BYTE	EIS9			
4BYTE	EIS9		Pa	
4BYTE	EIS9		Ohm	
4BYTE	EIS9		Ohm	
4BYTE	EIS9		Ohm m	
4BYTE	EIS9		H	
4BYTE	EIS9		sr	
4BYTE	EIS9		W/m_2	
4BYTE	EIS9		m/s	
4BYTE	EIS9		Pa	
4BYTE	EIS9		N/m	
4BYTE	EIS9		°C	
4BYTE	EIS9		K	
4BYTE	EIS9		J/K	
4BYTE	EIS9		W/mK	
4BYTE	EIS9		V/K	
4BYTE	EIS9		s	

Continued on next page . . .

Table 4.5.: KNX EIS data types

Data size	EIS Type	(S)igned / (U)nsigned	Unit	Meaning
4BYTE	EIS9		Nm	
4BYTE	EIS9		m_3	
4BYTE	EIS9		m_3/s	
4BYTE	EIS9		N	
4BYTE	EIS9		J	
2BYTE	EIS10	U		16-Bit unsigned
2BYTE	EIS10	S		16-Bit signed
4BYTE	EIS11	U		32-Bit unsigned
4BYTE	EIS11	S		32-Bit signed
4BYTE	EIS12			Access control
1BYTE	EIS13			ASCII Character
1BYTE	EIS14			ASCII Character
14BYTE	EIS15			14 * EIS13
10BYTE	EIS15			10 * EIS13 (unofficial Type)
14BYTE	EIS15			14 * EIS13 as BYTE ARRAY $(VT_UI1 \vert VT_ARRAY)$
8BYTE	EIS29		Wh	data type for power meter
8BYTE	EIS29		VARh	data type for power meter

Table 4.6.: Canonical Data types

Data size	EIS Type	(S)igned / (U)nsigned	Unit	Meaning
4BYTE	UI4			Unsigned Integer
3BYTE	UI4			Unsigned Integer only 3 of 4 bytes used
8BYTE	UI8			Unsigned Integer

! If you want to use one of these data types in a telegram, all indicated data type attributes must be entered in the telegram definition table:

Examples:

```
Correct:      Incorrect:
;2BYTE;EIS10;U;;  ;2BYTE;EIS10;;;
here the "Signed/Unsigned" attribute is missing
;1BIT;EIS1;;;   ;1BIT;EIS1;;°C;
EIS 1 does not hold any units
```

4.9.3. Modbus data types

Table 4.7.: Modbus data types

Modbus Type	Server Data Type	Size	Access Rights
discrete input	bool	fixed	Read-Only
coil	bool	fixed	Read-Write
input register	uint16	fixed	Read-Only
input register	int16	fixed	Read-Only
input register	uint32	fixed	Read-Only
input register	int32	fixed	Read-Only
input register	float	fixed	Read-Only
input register	double	fixed	Read-Only
input register	bool	fixed	Read-Only
input register	time	fixed	Read-Only
input register	string	1 – 246	Read-Only
input register	wstring	1 – 123	Read-Only
holding register	uint16	fixed	Read-Write
holding register	int16	fixed	Read-Write
holding register	uint32	fixed	Read-Write
holding register	int32	fixed	Read-Write
holding register	float	fixed	Read-Write
holding register	double	fixed	Read-Write
holding register	bool	fixed	Read-Write
holding register	time	fixed	Read-Write
holding register	string	1 – 246	Read-Write
holding register	wstring	1 – 123	Read-Write

4.9.4. BACnet object types

Table 4.8.: BACnet data types

BACnet Object Type	Server Data Type	Access Rights
Binary Input	bool	Read-Only
Binary Output	bool	Read-Write
Binary Value	bool	Read-Write
Analog Input	float	Read-Only
Analog Output	float	Read-Write
Analog Value	float	Read-Write
Accumulator	uint32	Read-Write
Pulse Converter	uint32	Read-Write
Multi-State Input	uint32	Read-Only
Multi-State Output	uint32	Read-Write
Multi-State Value	uint32	Read-Write
Life Safety Zone	uint32	Read-Only
Life Safety Point	uint32	Read-Only
Schedule	uint32, int32, string, float, double	Read-Only
Character String Value	string	Read-Write
Date Time Value	date	Read-Write
Integer Value	int32	Read-Write
Positive Integer Value	uint32	Read-Write
Large Analog Value	double	Read-Write
Lighting Output	float	Read-Write
Proprietary Objects	any	Read-Write or Read-Only

Table 4.9.: Possible BACnet Mapping

Server Data Type	Size	Unsigned Signed	Allowed BACnet types
BOOL	1 Bit		Binary Input Binary Output Binary Value
STR	Variable		Character String Value
CHAR	1 Byte	Signed	Analog Output Analog Input Analog Value Large Analog Value Integer Value
BYTE	1 Byte	Unsigned	Analog Output Analog Input Analog Value Accumulator Pulse Converter Multi-State Input Multi-State Output Multi-State Value Positive Integer Value
INT8	1 Byte	Signed	Analog Output Analog Input Analog Value Large Analog Value Integer Value
UINT8	1 Byte	Unsigned	Analog Output Analog Input Analog Value Large Analog Value Accumulator Pulse Converter Multi-State Input Multi-State Output Multi-State Value Positive Integer Value

Continued on next page . . .

Table 4.9.: Possible BACnet Mapping

Server Data Type	Size	Unsigned Signed	Allowed BACnet types
INT16	2 Byte	Signed	Analog Output Analog Input Analog Value Large Analog Value Integer Value
UINT16	2 Byte	Unsigned	Analog Output Analog Input Analog Value Large Analog Value Accumulator Pulse Converter Multi-State Input Multi-State Output Multi-State Value Positive Integer Value
INT32	4 Byte	Signed	Analog Output Analog Input Analog Value Large Analog Value Integer Value
UINT32	4 Byte	Unsigned	Analog Output Analog Input Analog Value Large Analog Value Accumulator Pulse Converter Multi-State Input Multi-State Output Multi-State Value Positive Integer Value
INT64	8 Byte	Signed	Analog Output Analog Input Analog Value Large Analog Value Integer Value
UINT64	8 Byte	Unsigned	Analog Output Analog Input Analog Value Large Analog Value Accumulator Pulse Converter Multi-State Input Multi-State Output Multi-State Value Positive Integer Value
FLOAT	4 Byte		Analog Output Analog Input Analog Value Large Analog Value
DOUBLE	8 Byte		Analog Output Analog Input Analog Value Large Analog Value
DATE			Date Time Value
TIME			Date Time Value

4.9.5. SNMP datapoint types

Table 4.10.: SNMP data types

SNMP Object Type	Server Data Type
Integer32	int32
Counter32	uint32
Gauge32	uint32
Counter64	uint64
IP	string
NetAddress	string
String	string

5. Client Configuration

To access the data points that are handled within the NETx BMS Server, different server interfaces that can be used by management clients (e.g. visualization clients) exist. NETx BMS Server 2.0 currently support the following server interfaces:

- OPC Data Access (OPC DA) 2.05
- VNET (proprietary network protocol developed by NETxAutomation Software GmbH)
- Web interface using HTTP
- BACnet/IP

5.1. Connection of OPC DA clients

Each OPC DA client uses its own way to select available of the OPC DA servers. Most of them provide a discovery mechanism that allows to search for available servers within the network. By providing a list of available server, the user can select the correct one. To connect to the NETx BMS Server 2.0 via an OPC DA connection, the OPC clients have to use the following connection string:

```
NETx.VOYAGER.SERVER.2.0
```

The NETx BMS Server can handle several OPC client connections at the same time in an independent way. Every connect or disconnect request of an OPC client is logged in the server logging facility (cf. Section 4.8.1).

5.2. Connection of VNET clients

If clients from NETxAutomation Software GmbH are used, it is possible to use the proprietary network protocol called VNET. VNET is based on TCP/IP and can be used as an alternative to OPC DA. Compared to the required DCOM settings needed by OPC DA, VNET is more easier to configure and to use. For using VNET over a Wide Area Network (WAN), only one TCP connection is required.

The IP address and the TCP port of the VNET interface of the NETx BMS Server is configured within the server configuration (cf. section 4.2.1). On the client side, only the IP address of the NETx BMS Server and the specified TCP port (default is 4530) have to be configured. Figure 5.1 shows an example how this is done within the NETx Voyager.

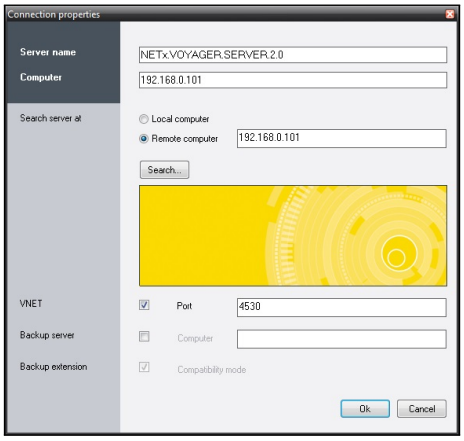


Figure 5.1.: VNET client config

5.3. Connection to NETx BMS Clients

A NETx BMS Client uses a standard web browser to access the NETx BMS Server via the integrated web server. Therefore, any device with a standard web browser that supports Javascript and HTML5 can act as a NETx BMS

Client.

5.3.1. Installation

As mentioned before, any device with a standard web browser with Javascript support can be used as a NETx BMS Client. Since the visualization is completely based on Javascript, no additional software or web browser plug-ins are necessary. Therefore, following web browser can be used without any further prerequisites:

- Internet Explorer
- Mozilla Firefox
- Google Chrome
- Opera
- Apple Safari
- ...

5.3.1.1. Configuration

Since no further software components than a standard web browser are necessary, no configuration is necessary. The only thing that has to be ensured is that Javascript is enabled within the web browser.

! Be aware about the risk of running a web server connected to public Internet without additional security measures. In addition, NETx BMS Clients shall not be configured with auto login, if they are connected to public networks.

5.3.1.2. Starting

To show a web based visualization project within the web browser, the correct HTTP URL has to be entered. This HTTP URL has the following format:

`http://<IP address of NETx BMS Server>:<Port>/<BMS client name>/`

`e.g.: http://192.168.0.1/RoomVisu/`

The IP address and the port can be configured within the server configuration of the NETx BMS Server (cf. Section 4.2.1). The NETx BMS Client name is the name of the client that is defined within the Project Tree (cf. Section 3.3.4).

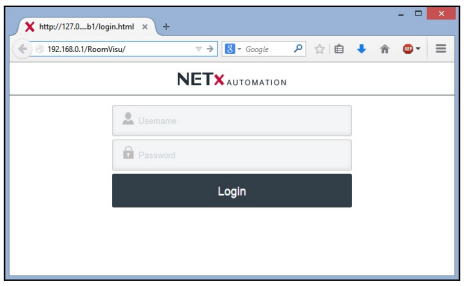


Figure 5.2.: NETx Web Voyager client

After having entered the correct HTTP URL, the login page of the NETx BMS Client visualization is shown. Here, the user name and password of the visualization project has to be specified. Figure 5.2 shows an example of the login screen of the NETx BMS Client.

5.4. NETx Touch client

In addition to use the standard web browser of iOS and Android devices for the web based visualization of the NETx BMS Server, a special app called NETx Touch is available, too. NETx Touch offers enhanced functionality

that cannot be provided by a standard web browser. In addition, web based visualizations can be integrated into a main/backup solution – if the main server is no longer available, the web based visualization will automatically switch to the backup server. NETx Touch also provides the opportunity to change the settings for the connection (address of the main and backup server, the name of the NETx BMS Client definition, user name and password) and to save it permanently protected from being changed by the end user. Furthermore, an automatic client discovery service simplifies the configuration. NETx Touch is available for iOS and Android and is free of charge.

Figure 5.3 shows an example of a web based visualization that is presented using NETx Touch.

Figure 5.3.: NETx BMS Client

5.4.1. Installation

NETx Touch is available within the Google Play Store and within Apple's iTunes. The App itself is for free.

5.4.1.1. Android

Open "Google Play Store", navigate to NETx Touch, and click it. Choose "Install", confirm the download and the program will be installed immediately.

5.4.1.2. iOS

Open the "App Store", navigate to NETx Touch, and click it. Hit the "free" button and choose "Install". The app will be installed immediately.

5.4.2. Settings

Once the NETx Touch is started the settings menu can be reached by touching the icon in the lower right corner of the application. The following section describes each entry and its influence to the app.

5.4.2.1. General Settings

Settings Protection

Parameter:	Enabled
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	If this parameter is turned on, the settings can be entered only by entering the correct password.

Parameter:	Password
Scope:	String
Default value:	None
Unit:	None
Description:	This contains the password the settings are protected with. It must be at least 3 letters long. It can (and should) contain capitalized letters, small letters, numbers, and special characters. White spaces such as line-feeds or tabulators are not allowed.

Parameter:	Reenter
Scope:	String
Default value:	None
Unit:	None
Description:	Enter the password again in this field to confirm it.

User Interface

Parameter:	Lock Rotation
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	This locks the rotation feature of the application. Nevertheless, if the device is turned upside-down the application will change its orientation accordingly.

Parameter:	Allow Zoom
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	This allows or locks the to zoom possibility inside the app. It might be of use in some cases but it is recommended to leave it off.

Parameter:	Hardware acceleration (only Android version)
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	The option activates or deactivates the hardware acceleration of the Android device which finally improves the performance of the application.

Parameter:	Delete App from Startbutton (only Android version)
Scope:	
Default value:	None
Unit:	None
Description:	By selecting this option, the launcher button of the Android device can be reconfigured. For special applications like hotel rooms it can be necessary to hinder user from leaving the app.

Parameter:	Clear web cache
Scope:	
Default value:	None
Unit:	None
Description:	If the option is turned on, the current webcache of the application is cleared when the settings menu is closed.

Demomode

Parameter:	Enabled
Scope:	ON/OFF
Default value:	ON
Unit:	None
Description:	This option activates the demo mode of the application. If the connection to the NETx BMS Server should be established, this option has to be false.

Main Server

Parameter:	Host
Scope:	String
Default value:	None
Unit:	None
Description:	Enter the IP address or host name of the main server in here.

Parameter:	Port
Scope:	Integer
Default value:	None
Unit:	None
Description:	Enter the port of the host in here.

Parameter:	Timeout (s)
Scope:	Integer
Default value:	None
Unit:	None
Description:	This parameter contains the time for the connection to timeout in counted in seconds.

Backup Server (Optional)

The NETx Touch is able to connect to act in a main-backup-solution. In case the main server cannot be reached the app will attempt to connect to the backup server and start its work there. “Port” and “Timeout” will be taken from the “Main Server” definitions.

Parameter:	Enabled
Scope:	ON/OFF
Default value:	OFF
Unit:	None
Description:	If this parameter is turned on, the app will try to connect to the backup server in case the main server cannot be reached.

Parameter:	Host
Scope:	String
Default value:	None
Unit:	None
Description:	Enter the IP address or host name of the backup host in here.

BMS Client

This part is responsible for the log-in information of a client definition. What client shall the app connect to? What user information shall be provided to the server? This user information will be provided to the backup server also. Thus the defined user information on main- and backup-visualization have to be identical.

Parameter:	Client
Scope:	String
Default value:	None
Unit:	None
Description:	The content of this field represents the name of the BMS Client defined in the server. It needs to match all letters and their cases.

Parameter:	User
Scope:	String
Default value:	None
Unit:	None
Description:	Put in the user name defined in the visualization project.

Parameter:	Password
Scope:	String
Default value:	None
Unit:	None
Description:	Put in the password defined in the visualization project.

Option:	Discover Clients
Purpose:	Starts the client discovery
Description:	In case the server provides the service it is possible to get the defined client names off of the server. When the button is pressed a list will appear which contains the current client definitions of the server. By selecting one the user is lead back into the main screen of the settings where the content of “Client” will now show the chosen name.

5.4.2.2. Special Settings

iOS

On an iOS device the NETx Touch offers an additional feature in the application settings not of the app itself but inside the general settings for the NETx Touch.

Parameter:	Enable Settings
Scope:	ON/OFF
Default value:	ON
Unit:	None
Description:	If this option is turned off the application settings are not available at during run-time of the app. The settings button in the lower right corner will not be shown and thus the settings are hidden to the user. The settings can be altered only if this general setting is turned to “ON” again.

A. Appendix

A.1. Acronyms

BACnet Building Automation and Control Networking Protocol
COV Change of Value
DSN Data Source Name
ESF EIB Session File
F False
HTTP HyperText Transfer Protocol
JSON JavaScript Object Notation
MaRS Metering and Reporting System
MIB SNMP Management Information Base
OID SNMP Object Identifier
OPC DA OPC Data Access
SNMP Simple Network Management Protocol
T True
USB Universal Serial Bus
VIL Virtual Item Link
WAN Wide Area Network

A.2. Licensing

Two different possibilites are available:

- Hardlock – Hardware based (Universal Serial Bus (USB) Dongle) security system
- Softlock – Software based security system

A.2.1. Hardlock

One of the solutions is hardlock. Just insert the USB-Dongle into any USB port in your system and let the hardware driver get installed. The software will automatically recognize and read your license from the USB dongle. It might be necessary to restart the software to make it recognize the license.

! The USB dongle has to be connected at all times from start of the application on. If it is disconnected the correct work of the software will stop after two warnings (which is approximately after 15 minutes).

A.2.2. Softlock

In this case the software license is assigned to a local code that depends on a checksum of the local hard- and software.

! The local code of the soft lock license can be changed by any major change of hardware (e.g. network adapter) or software (e.g. operating system). After such change it might be necessary to license the software again.

A.2.2.1. Software Licensing

The licensing process can be initiated by starting the “License Manager” from the Windows start menu (“NETxAutomation”, “NETx BMS Server 2.0”, “NETx Registration”).

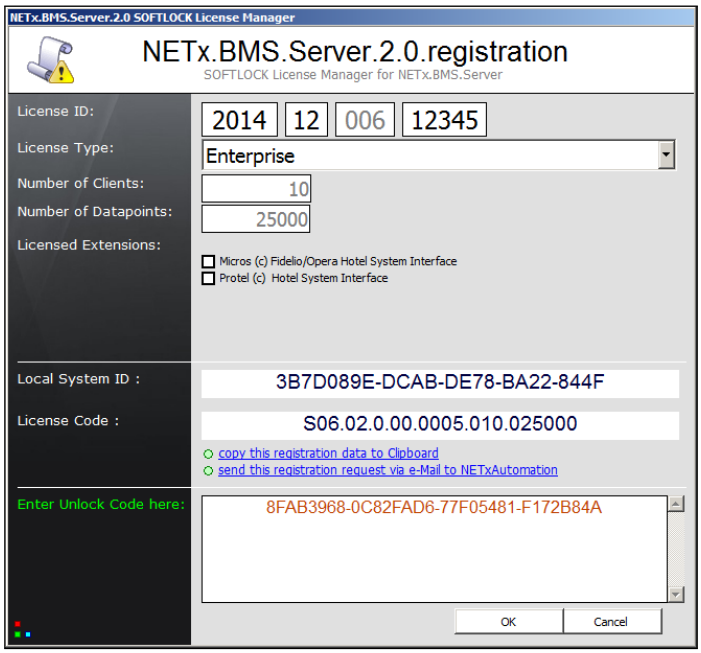


Figure A.1.: Licensing Software

The License Manager will show up with several fields:

- License ID – This block consists of 4 fields. Please fill in the License ID from the delivered invoice.

- License Type – Select the type according to the delivered invoice here. Amount of clients and amount of datapoints will depend on it. If none of the predefined licenses matches select “Custom” and enter the amount of clients and datapoints.
- Number of Clients – This field defines the number of clients to be licensed. For custom licenses, please enter the amount of clients here. If the order was a predefined package this field is not editable but will show the according number of clients.
- Number of Datapoints – This field defines the number of datapoints to be licensed. For custom licenses, please enter the amount of datapoints here. If the ordered was a predefined package this field is not editable but will show the according number of datapoints.
- Licensed Extensions – Check the extensions which have been ordered in here.
- Local System ID – This is the local system key. It is automatically generated by the “License Manager”.
- License Code – This code contains most of the information above. It is generated automatically.

Once the above listed fields are all filled out with the correct information send an email directly to NETxAutomation Software GmbH by clicking the according link. If there is no access to the Internet, choose the link “copy to clipboard”. Open a text file in which the clipboard content needs to be inserted (usually via [Ctrl]+[V] or “Edit” and “Paste”). Now send it from any other computer by mail to register@netxautomation.com.

The following entries must be sent to license the software:

- License ID
- License Type
- License Code
- Local Code
- Date
- Software Version

Transmit this data to NETxAutomation Software GmbH to receive the “Unlock Code”. Once it was received the unlock code must be inserted into the bottom field and confirmed with “OK”. It is to be made sure to copy the “Unlock Code” without any blanks, tabs or other white spaces. If an error message is received after the process, check the given data and try the registration once more.

! The user executing the licensing software must be local administrator. The program needs to be started as administrator.

A.2.2.2. Move a license

When it is planned to move the NETx BMS Server 2.0 from one system to another or the system will change (hardware upgrade), it is useful to unlicense the software. The unlicensing process can be initiated by starting the “License Manager ...” from the Windows start menu (“NETxAutomation”, “NETx BMS Server 2.0”, “NETx Registration”)

Press the “Transfer/Remove License” button to gain the removal code. Send this code to NETxAutomation Software GmbH to receive the new “Unlock Code” to license the software on the new or upgraded system.

A.2.3. License Count

The NETx BMS Server is licensed after datapoints and BMS Clients. In addition to regular(physical) datapoints the license includes always an equally high amount of virtual datapoints. Datapoints are all bus layer objects defined in tables and listed either in the XIO branch of the Item Tree or inside the Aliases. Virtual datapoints are considered all server items which have no direct relation to a physical device. They are either calculated items or items with a value brought in by an alien software (e.g. NETx OPC Bridge). Not all virtual items count into the license limit.

The following three lists define exactly which items count where:

Datapoints (regular/physical)

- KNX Items

- BACnet Items – only the BACnet Object itself is counted as one item. The items underneath representing the priorities are not counted.
- Modbus Items
- JSON Items
- Aliases – since items with an alias are rather moved into this branch than copied, the aliases will count but original items remain uncounted.

Virtual Datapoints (counted)

- All additional items created by in-/out-conversion
- All items in the branch NETx\Custom (added by LUA nxa.AddCustomItem())
- All items in the branch NETx\Module
- All items in the branch NETx\VIRTUAL (added by definition in the table)
- All items in the branch NETx\XCOMMAND
- All items in the branch NETx\XCON

Virtual Datapoints (not counted)

- All items in the branch NETx\API (e.g. the Fidelio interface)
- All items in the branch NETx\Server
- All items in the branch NETx\Today
- All items in the branch NETx\Geo
- All items in the branch NETx\VAR – all predefined variables

A.3. Support and contact

Please send all your support questions to:
support@NETxAutomation.com
If you have general questions regarding the product and service please send your email to:
office@NETxAutomation.com

A.4. System Requirements

A.4.1. Hardware

- Processor: Intel or AMD 1.6GHz (Multicore recommended)
- System Memory: 2048MB
- Harddisk Space: 8GB (16GB recommended)
- Network Adapter: 100 MBit/s
- Screen Resolution: 1280 x 1024 Pixel (for BMS Studio)

A.4.2. Supported Operating Systems

- Microsoft Windows 7 32bit Servicepack 1
- Microsoft Windows 7 64bit Servicepack 1
- Microsoft Windows 8 64bit
- Microsoft Windows 8.1 64bit
- Microsoft Windows Server 2008 32bit Servicepack 2
- Microsoft Windows Server 2008 64bit Servicepack 2
- Microsoft Windows Server 2008 Release 2 64bit Servicepack 1
- Microsoft Windows Server 2012 64bit
- Microsoft Windows Server 2012 R2 64bit

A.4.3. Other

- .NET Framework: 3.5 (included in setup)
- Microsoft Visual Studio Redistributable 2008 32 or 64 bit (included in setup)
- Microsoft Visual Studio Redistributable 2010 32 bit (included in setup)
- Microsoft Access Redistributable 2010 32 bit (included in setup)