★★★

Basic / Advanced / Expert

Application note

spaceLYnk as a Modbus slave

How to set spaceLYnk to act as a Modbus TCP or Modbus RTU slave.



Schneider Electric

Safety Information

Important Information

Read these instructions carefully before trying to install, configure, or operate this software. The following special messages may appear throughout this bulletin or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.



The addition of either symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.



This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠DANGER
DANGER indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

⚠WARNING
WARNING indicates a potentially hazardous situation which, if not avoided, can result in death or serious injury.

⚠CAUTION
CAUTION indicates a potentially hazardous situation which, if not avoided, can result in minor or moderate injury.

NOTICE
NOTICE is used to address practices not related to physical injury. The safety alert symbol shall not be used with this signal word.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction, installation, and operation of electrical equipment and has received safety training to recognize and avoid the hazards involved.

Safety Precautions

⚠WARNING

HAZARD OF INCORRECT INFORMATION

- Do not incorrectly configure the software, as this can lead to incorrect reports and/or data results.
- Do not base your maintenance or service actions solely on messages and information displayed by the software.
- Do not rely solely on software messages and reports to determine if the system is functioning correctly or meeting all applicable standards and requirements.
- Consider the implications of unanticipated transmission delays or failures of communications links.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information that is contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2014 Schneider Electric. All rights reserved

Table of Contents

1 Introduction 6

2 Design 7

3 Configuration 8

3.1 Modbus RTU slave 8

3.1.1 Modbus slave proxy server functions 8

3.1.2 Modbus slave handler 10

3.1.3 Event script for register update 12

3.2 Modbus TCP slave 13

4 Example - communication with Com'X 200 14

4.1 Setting of spaceLYnk 14

4.2 Setting of Com'X 200 15

4.3 Functional test 17

5 Conclusion 18

6 Appendix 18

6.1 Glossary 18

6.2 Reference 18

1 Introduction

This application note describes how to configure spaceLYnk as a Modbus slave. This can be very useful when there is a need to read data from spaceLYnk, but BACnet or KNX network cannot be used. There can be some systems like SCADA or some HVAC control systems, which need to read data from KNX installation, but the systems do not support the KNX communication. In this case they can read the data over Modbus network, where spaceLYnk acts as a Modbus slave device.

A glossary is available in the appendix chapter of this document. Please refer to it whenever necessary.

Use case example 1:

Existing KNX installation needs to be interconnect with heating system managed by an Xenta 701. Xenta controller can read data from KNX devices over Modbus protocol and use it as control parameters.

Use case example 2:

Share energy metering, room occupancy, light level and other energy relevant data with Energy Server Com'X200. Link KNX objects with Modbus registers and let Com'X200 read the values from spaceLYnk and display data collected from KNX installation as a part of energy dashboard.

Competencies

This document is intended for readers who have been trained in homeLYnk, spaceLYnk products. The integration should not be attempted by someone who is new in the installation of either products. In addition we recommend that you should be familiar with KNX communication, Modbus protocol and advanced in Lua scripting.

System prerequisities

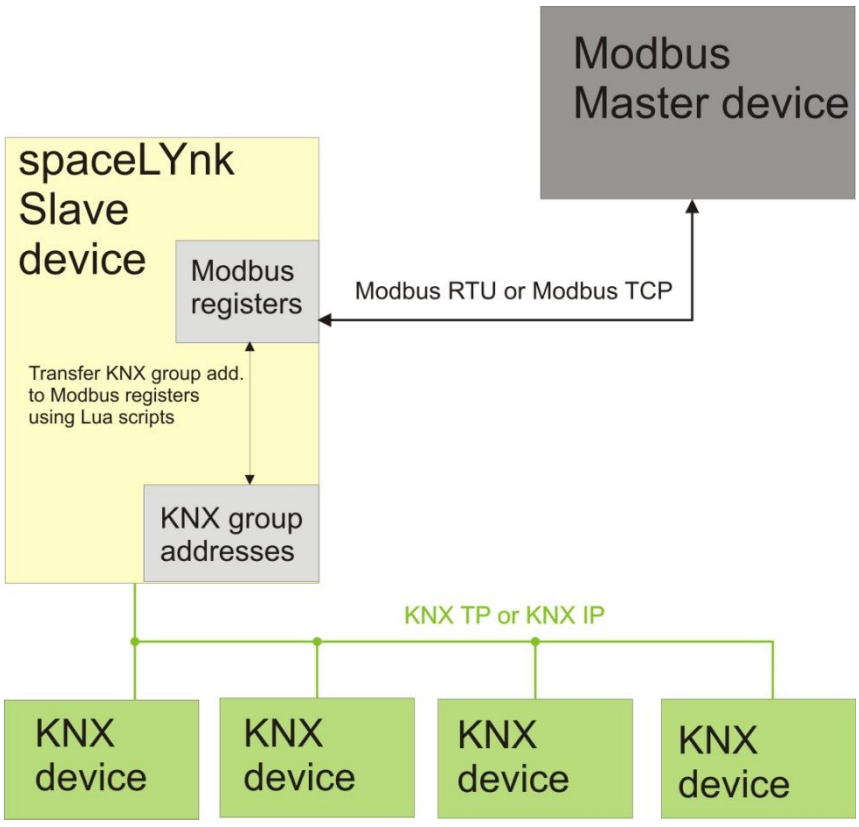
Software	Version	Download
spaceLYnk	2.1 and newer	http://www.schneider-electric.com

Table 1: software versions of used software

2 Design

In order to better understand the communication between KNX devices and Modbus master device please refer to the scheme in Picture 1.

SpaceLYnk is collecting data from KNX devices and store them in its own objects. All objects in spaceLYnk correspond to some KNX group addresses. Once the spaceLYnk is set as a Modbus slave then Modbus registers are created according to the mapping (Chapter 3.1.2). Modbus registers are updated in spaceLYnk using values from KNX objects or some other values internally calculated in spaceLYnk using Lua scripts. Modbus master device can access Modbus registers in spaceLYnk, read them and change theirs values. This is indirect way how Modbus master device communicates with KNX devices, even if Modbus master device itself does not support KNX communication.



Picture 1: Scheme of master-slave communication

3 Configuration

Configuration of Modbus slave is split into two subchapters depending on protocol version you choose:

1. Chapter 3.1 Modbus RTU slave
2. Chapter 3.2 Modbus TCP slave

3.1 Modbus RTU slave

Configuration of Modbus RTU slave is done using instructions and scripts in 3 subchapters below.

Main configuration steps in following subchapters:

1. **Modbus slave proxy server function**
   Add prepared functions into *Configurator >> Scripting >> Common function*.
2. **Modbus slave handler**
   Create resident script called ”handler”. Script example is included below.
3. **Event script for register update**
   Create event scripts, which update Modbus registers according to changes of KNX group addresses.

3.1.1 Modbus slave proxy server functions

Step 1 Go to *Configurator >> Scripting >> Common function*.

Step 2 Append (copy-paste) the script below into Common function script.

```
-- modbus proxy
mbproxy = {
  -- supported function list
  functions = {
    'readdo',
    'readcoils',
    'readdi',
    'readdiscreteinputs',
    'readao',
    'readregisters',
    'readai',
    'readinputregisters',
    'writebits',
    'writemultiplebits',
    'writeregisters',
    'writemultipleregisters',
    'reportslaveid',
    'getcoils',
    'getdiscreteinputs',
    'getinputregisters',
    'getregisters',
    'setcoils',
    'setdiscreteinputs',
    'setinputregisters',
    'setregisters',
  },
  -- new connecton init
  new = function()
    require('rpc')
    local mb = setmetatable({}, { __index = mbproxy })
    mb.slaveid = 0
    mb.rpc = rpc.client('127.0.0.1', 28002, 'mbproxy')
    for _, fn in ipairs(mbproxy.functions) do
      mb[ fn ] = function(self, ...)
        return mb:request(fn, ...)
      end
    end
    return mb
  end
}
-- set local slave id
function mbproxy:setslave(slaveid)
  self.slaveid = slaveid
end
-- send rpc request for a spefic function
function mbproxy:request(fn, ...)
  local res, err = self.rpc:request({
    fn = fn,
    params = { ... },
    slaveid = self.slaveid or 0,
  })
  -- request error
  if err then
    return nil, err
  -- request ok
  else
    -- reply with an error
    if res[ 1 ] == nil then
      return nil, res[ 2 ]
    -- normal reply
    else
      return unpack(res)
    end
  end
end
```

Common functions – Modbus slave proxy server functions

Step 3 Click **Save and Close** in bottom left.

3.1.2 Modbus slave handler

Create a resident script called “handler” with sleep time=0. Handler script example shown below can be used as a template for your handler.

Step 1 Go to *Configurator >> Scripting >> Resident >> Add new script.*

Step 2 Set the Sleep interval to 0. And check the Active option.



Step 3 Click **save** button.

Step 4 Open script editor.

Step 5 Use the code from example below. Copy and paste it into the resident script.

Step 6 Adjust your Modbus slave settings in 5 steps by modifying the code of script example above:

a) Set serial communication baudrate, parity, data bits, stop bits and duplex

```
mb:open('/dev/RS485', 19200, 'E', 8, 1, 'H')
```

b) Set slave-id of the spaceLYnk.

```
mb:setslave(10)
```

c) Set Modbus registers mapping.

Number of registers for coils, inputs, holding registers and input registers.

```
mb:setmapping(10, 10, 10, 10)
```

d) Set callback function which is executed for each coil write.

```
mb:setwritecoilcb(function(coil, value)...
```

If master device write value to spaceLYnk coil register, code of callback function is executed. For example coil number and its value is printed into log (see the example below):

```
mb:setwritecoilcb(function(coil, value)
    log('coil: %d = %s', coil, tostring(value))
end)
```

e) Set callback function which is executed for each register write.

```
mb:setwriteregistercb(function(coil, value)...
```

If master device write value to spaceLYnk holding register, code of callback function is executed. For example, if value of register 2 is changed, new value is saved into specified group address (see the example below):

```
mb:setwriteregistercb(function(register, value)
    If register==2 then
    grp.write('1/1/1',value)
    end
end)
```

```
-- modbus init
if not mb then
  --*****************************************************
  -->>>>>>>>>>Modbus slave setting<<<<<<<<<<<<<<<<<<<<<<<<
  require('luamodbus')
  mb = luamodbus.rtu()
  mb:open('/dev/RS485', 19200, 'E', 8, 1, 'H')
  mb:connect()
  -- slave id
  mb:setslave(10)
  -- init slave storage for coils, discrete inputs, holding registers and input registers
  mb:setmapping(10, 10, 10, 10)
  --*****************************************************
  -- >>>>>>>>>>>Coil write callback<<<<<<<<<<<<<<<<<<<<<<
  mb:setwritecoilcb(function(coil, value)
    if coil == 0 then
      grp.write('1/1/1', value, dt.bool)
    else
      alert('coil: %d = %s', coil, tostring(value))
    end
  end)
  --*****************************************************
  -->>>>>>>>>>>Register write callback<<<<<<<<<<<<<<<<<<<
  mb:setwriteregistercb(function(register, value)
    if register == 0 then
      -- send value limited to 0..100
      grp.write('4/1/5', math.min(100, value), dt.scale)
    else
      alert('register: %d = %d', register, value)
    end
  end)
  --*****************************************************
end


-- server part init
if not server then
  require('rpc')
  -- incoming data handler
  local handler = function(request)
    local fn, res
    fn = tostring(request.fn)
    if not mb[ fn ] then
      return { nil, 'unknown function ' .. fn }
    end
    if type(request.params) == 'table' then
      table.insert(request.params, 1, mb)
      res = { mb[ fn ](unpack(request.params)) }
    else
      res = { mb[ fn ](mb) }
    end
    return res
  end
  server = rpc.server('127.0.0.1', 28002, 'mbproxy', handler, 0.01)
end
mb:handleslave()
server:step()
```

Modbus slave handler script example – Modbus RTU

Step 7 Click **Save and Close** in bottom left.

For more detailed description of Modbus functions used in this Modbus handler script please refer to spaceLYnk User Guide.

Event script for register update

Registers are created in spaceLYnk and can be read over Modbus RTU. If there is change of register made by master device, spaceLYnk can react on this change performing the code of callback functions (chapter 3.1.2). Update of Modbus registers in spaceLYnk is not performed with change of KNX group addresses. Use event scripts in order to update Modbus registers values with KNX group address change. See the examples below.

Warning:

Because there is a real risk of spaceLYnk overload, it is better to update Modbus registers when KNX group addresses are changed (not periodically). Use event scripts for particular group addresses, do not use update in resident script.

Example 1:

- Event script which changes Modbus slave coil register.
- It must be mapped to a group address with binary value.
- Value – current value of event trigger value
- Command *mb:setcoils(0, value)* set coil register 0 equal to value.

```
value = event.getvalue()
mb = mbproxy.new()
mb:setcoils(0, value)
```

Event script example 1

Example 2:

- Example event script which changes Modbus slave register (register address 5).
- It must be mapped to a group address with 1byte or 2byte value.
- Value – current value of event trigger value
- Command *mb:setregisters(5, value)* set modbus register 5 equal to value.

```
value = event.getvalue()
mb = mbproxy.new()
mb:setregisters(5, value)
```

Event script example 2

3.2 Modbus TCP slave

In case of Modbus TCP communication, the configuration procedure is almost the same as Modbus RTU. The only difference is in Modbus slave handler script, where TCP connection needs to be used instead of RTU. Command *mb:open(address,port)* defines IP address of slave device (the same as IP address of spaceLYnk). The modified part is highlighted in the script example below (green).

```
-- modbus init
if not mb then
  --*****************************************************
  -->>>>>>>>>>Modbus slave setting<<<<<<<<<<<<<<<<<<<<<<<
  require('luamodbus')
  mb = luamodbus.tcp()
  -- IP:10.154.12.73, port: 502
  mb:open('10.154.12.73', 502)
  mb:connect()
  -- slave id
  mb:setslave(10)
  -- init slave storage for coils, discrete inputs, holding registers and input registers
  mb:setmapping(10, 10, 10, 10)
  --*****************************************************
  -- >>>>>>>>>>>Coil write callback<<<<<<<<<<<<<<<<<<<<<<<
  mb:setwritecoilcb(function(coil, value)
    if coil == 0 then
      grp.write('1/1/1', value, dt.bool)
    else
      alert('coil: %d = %s', coil, tostring(value))
    end
  end)
  --*****************************************************
  -->>>>>>>>>>>>Register write callback<<<<<<<<<<<<<<<<<<<
  mb:setwriteregistercb(function(register, value)
    if register == 0 then
      -- send value limited to 0..100
      grp.write('4/1/5', math.min(100, value), dt.scale)
    else
      alert('register: %d = %d', register, value)
    end
  end)
  --*****************************************************
end

-- server part init
if not server then
  require('rpc')
  -- incoming data handler
  local handler = function(request)
    local fn, res
    fn = tostring(request.fn)
    if not mb[ fn ] then
      return { nil, 'unknown function ' .. fn }
    end
    if type(request.params) == 'table' then
      table.insert(request.params, 1, mb)
      res = { mb[ fn ](unpack(request.params)) }
    else
      res = { mb[ fn ](mb) }
    end
    return res
  end
  server = rpc.server('127.0.0.1', 28002, 'mbproxy', handler, 0.01)
end
mb:handleslave()
server:step()
```

Modbus slave handler script example – Modbus TCP

4 Example - communication with Com'X 200

Energy server Com'X 200 collects and stores power metering and environmental data. It is mainly used to read data from number of various power meters. In some cases it can be useful to read and track environmental variables (temperature, occupancy, air humidity, …) as well. This information gives more complex overview of the systems and helps to improve energy efficiency of the system. Such an environmental data are easy to read from KNX network. SpaceLYnk as a Modbus slave device can provide all information read from KNX devices to Com'X 200.

In following subchapters you will find description how to set Modbus communication between spaceLYnk and Com'X 200.

4.1 Setting of spaceLYnk

Step 1 Set spaceLYnk as a Modbus TCP slave using procedures from previous chapters.

Step 2 Create new objects in spaceLYnk.

Group address	Object name	Data type
5/5/1	Temperature Room1	9.001 Temperature
5/5/2	Humidity Room1	5.001 Scale

Step 3 Link object to Modbus registers using event scripts.

Event script for group address 5/5/1 (Temperature):

```
value = event.getvalue()
mb = mbproxy.new()
mb:setregisters(3, value)
```

When value of 5/5/1 is changed, the value is saved to Modbus holding register 3.

Event script for group address 5/5/2 (Humidity):

```
value = event.getvalue()
mb = mbproxy.new()
mb:setregisters(4, value)
```

When value of 5/5/2 is changed, the value is saved to Modbus holding register 4.

SpaceLYnk with all the setting can be restored using backup file: **AN016_spaceLYnk_backup.tar.gz.**

4.2 Setting of Com'X 200

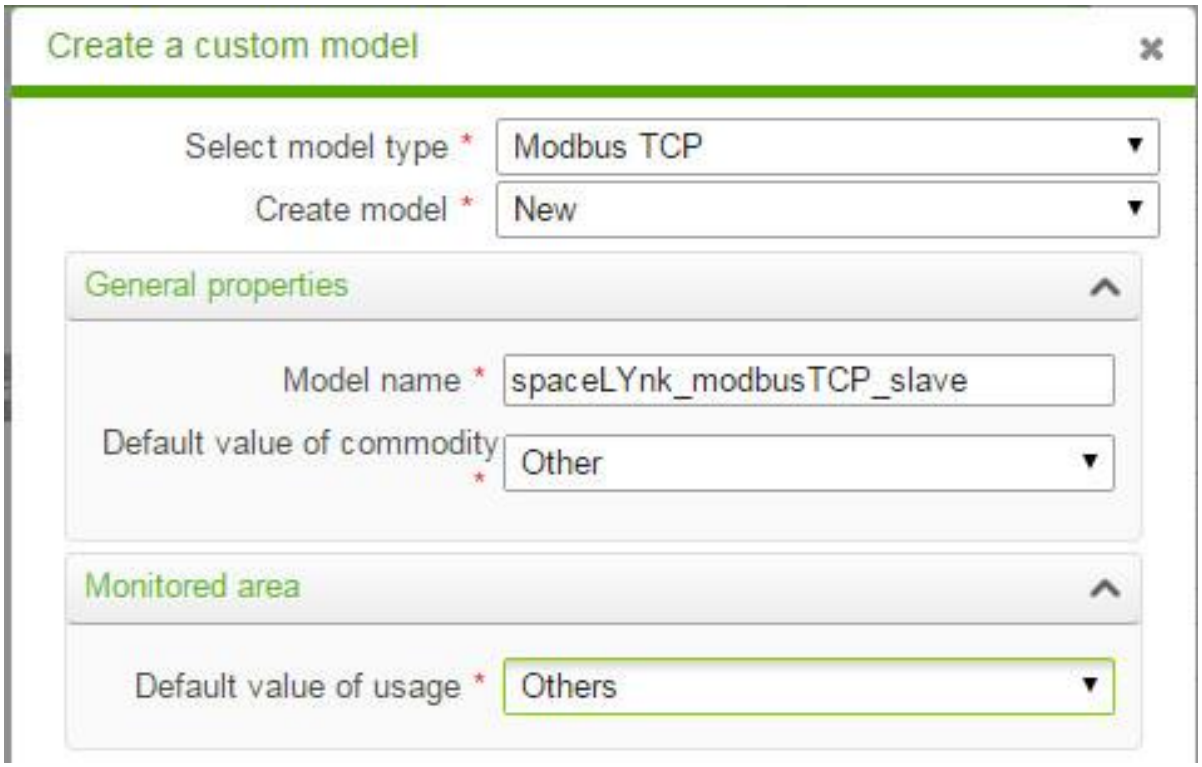
Step 1 Open the web interface of Com'X 200 in your browser.

Step 2 Navigate to table *Custom Library.*

Step 3 Click **+ Create** button in bottom-left of the screen



Step 4 Create a custom model as shown in the picture below and click **Create** button.



Step 5 Select *Custom Library >> Model list >> Modbus TCP >> homeLYnk_modbusTCP_slave*

Step 6 Click button **New frame**.

Step 7 Click **Edit frame** button and add two items as it is shown in picture below.



Step 8 Click button **Save changes** in right-bottom.

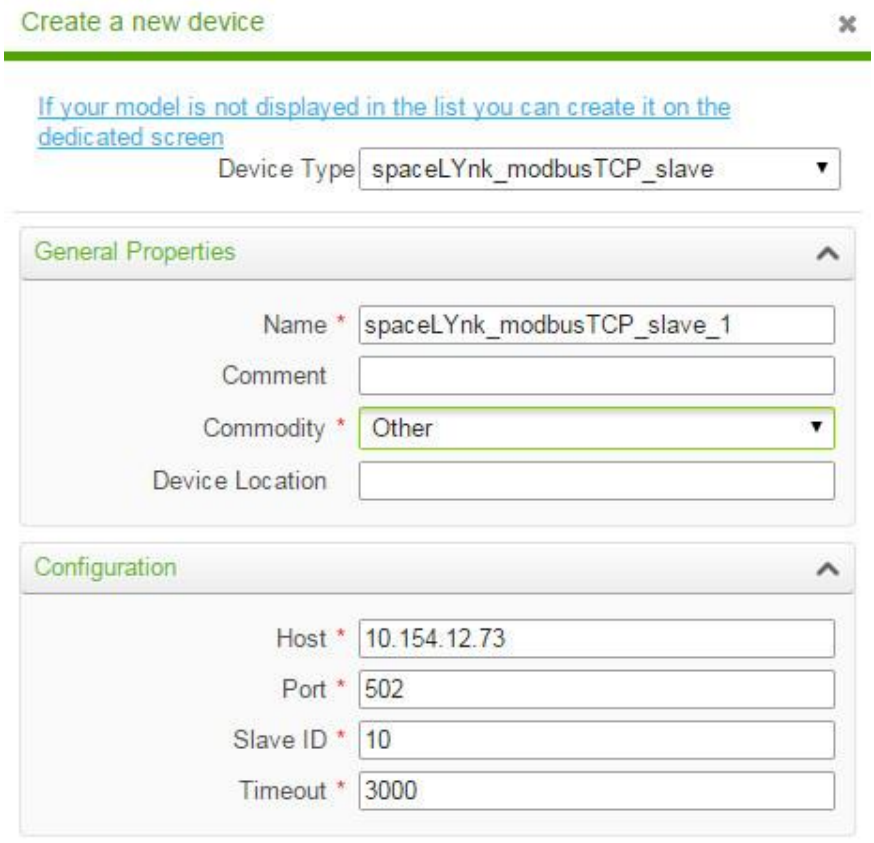
Step 9 Navigate to *Device Settings >> Ethernet >> Device.*

In dropdown menu select *Create new device.*



Step 10 Open dropdown menu and set device type to spaceLYnk_modbusTCP_slave (this device type has been created in Step 4).

Step 11 Set Commodity, Host (IP address of spaceLYnk), Port (Modbus port set in spaceLYnk), Slave ID and Timeout.



Step 12 Click **Create** button.

Step 13 Navigate to *Device Settings >> ComX200 >> Ethernet >> spaceLY_modbusTCP_slave*

Step 14 In *Measurement table* you see the values read from spaceLYnk. Check the option “Publish”.

Measurements Table:

Measurement name	Value	Unit	Publish
Temperature	26	°C	☑
Humidity Rate	82	%	☑

Step 15 Navigate to *Measurrement Table* and check the option *“Other”.*

Step 16 See the table with values read from spaceLYnk.

spaceLYnk_modbusTCP_slave

spaceLYnk_modbusTCP_slave_1

Lighting

Building
Floor
Zone

Temperature	26	°C
Humidity Rate	82	%

4.3 Functional test

Step 1 Change values of objects 5/5/1 and 5/5/2 in spaceLYnk.

Step 2 Check that values has been changed in Com’X 200 as well.

5 Conclusion

This document describes step-by-step setting of spaceLYnk as a Modbus slave device. The main benefit of setting spaceLYnk as Modbus slave is interconnection between KNX installation and other systems, which does not support KNX communication standard. See the usage examples mentioned in Chapter 1.

In order to check all functionality of Modbus slave it is useful to configure second spaceLYnk/homeLYnk as a Modbus master, which reads registers from Modbus slave.

There is no special focus on Modbus protocol, Lua scripting nor basic use of spaceLYnk GUI. If you need some more details about topic, which is not deeply described here, please refer to the documents listed in References.

6 Appendix

6.1 Glossary

The following table describes the acronyms and defines the specific terms used in this document.

Abbreviation	Description
GUI	Graphical User Interface
HVAC	Heating, Ventilation and Air Conditioning.
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
TCP	Transmission Control Protocol

Table 2: specific terms

6.2 Reference

Document title	Reference
spaceLYnk User Guide	www.schneider-electric.com
Modbus Serial Line Planning and Installation Guide	www.schneider-electric.com
Modbus organization official website	www.modbus.org

Table 3: reference

Schneider Electric Industries SAS

Head Office
35, rue Joseph Monier
92506 Rueil-Malmaison Cedex
FRANCE

www.schneider-electric.com