MicroSCADA Pro SYS 600 9.3
Modbus Slave Protocol
Contents

1 Copyrights ................................................................. 5

2 Introduction .............................................................. 7

2.1 This manual .......................................................... 7
2.2 Related documents .................................................. 8
2.3 Document revisions .................................................. 8

3 Safety information ....................................................... 9

3.1 Backup copies ......................................................... 9
3.2 Fatal errors ........................................................... 9

4 Instructions ............................................................... 11

4.1 General ................................................................. 11
4.2 Configuration ........................................................ 11
4.2.1 Base system configuration ..................................... 12
4.2.2 Communication system configuration ..................... 14
4.3 After configuration .................................................. 21
4.4 Start-up ................................................................. 22
4.5 How to test the configuration ..................................... 25

5 Technical description ................................................ 27

5.1 Modbus protocol ..................................................... 27
5.2 Communication ....................................................... 27
5.2.1 CPI application program ....................................... 27
5.2.2 Connection to a SYS 600 base system ..................... 27
5.2.3 Data flow .......................................................... 28
5.2.4 Addressing schematics with Modbus protocol .......... 29
5.2.5 Device communication attributes ............................ 31
5.3 Command procedures .............................................. 35
5.3.1 Command procedures in COM 500/......................... 35
5.3.2 Command procedures in SYS 600 ......................... 37
5.3.2.1 Command procedures for process data ............... 37
5.3.2.2 Command procedures for commands ................. 40
5.4 Function codes ....................................................... 41
5.4.1 Function codes for process data ............................ 43
5.4.1.1 Function code 01 - Read coil status .................. 43
5.4.1.2 Function code 02 - Read input status ............... 44
5.4.1.3 Function code 03 - Read holding registers ......... 44
5.4.1.4 Function code 04 - Read input registers .......... 45
5.4.2 Function codes for commands ............................... 45
5.4.2.1 Function code 05 - Forcing a coil .................... 45
5.4.2.2 Function code 06 - Modify register content .... 45
5.4.2.3 Function code 08 - Diagnostics ................. 46
5.4.2.4 Function code 11 - Fetch communication
event counter ................................................... 48
5.4.2.5 Function code 15 - Force multiple coils ...... 49
5.4.2.6 Function code 16 - Modify multiple registers . 49

5.5 Status codes ................................................................................. 49

5.6 Interoperability list ....................................................................... 50
5.6.1 Network Configurations ....................................................... 50
5.6.2 Physical Layer ........................................................................... 50
5.6.3 Link Layer ............................................................................... 51
5.6.4 Transmission Speed (only serial) ................................. 51
5.6.5 Transmission Settings (only serial) ......................... 51
5.6.6 Application Layer ................................................................. 51

Index ........................................................................................................ 53
1 Copyrights

The information in this document is subject to change without notice and should not be construed as a commitment by ABB Oy. ABB Oy assumes no responsibility for any errors that may appear in this document.

In no event shall ABB Oy be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB Oy be liable for incidental or consequential damages arising from the use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB Oy, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

Copyright © 2013 ABB Oy. All rights reserved.

Trademarks

ABB is a registered trademark of ABB Group. All other brand or product names mentioned in this document may be trademarks or registered trademarks of their respective holders.

Guarantee

Please inquire about the terms of guarantee from your nearest ABB representative.

Third Party Copyright Notices

This software uses pugixml library (http://pugixml.org). pugixml is Copyright © 2006-2013 Arseny Kapoulkine.

This software uses 7-Zip program (http://www.7-zip.org). 7-Zip is licensed under the GNU LGPL license.
2 Introduction

2.1 This manual

This manual provides thorough information on the Modbus slave protocol and information related to it. It describes how to configure the base system and the Modbus slave application for communication between them and with the Modbus master.

In addition to this configuration, the base system needs to be configured for the process communication. For information about this subject, see other manuals, for example SYS 600 Application Objects or SYS 600 COM 500i User’s Guide. The Modbus master needs to be configured as well.

Modbus slave

The Modbus slave protocol is mainly used for upper level communication between SYS 600 and a network control system as illustrated by Figure 2.1. In MicroSCADA, the Modbus slave protocol is implemented using the CPI protocol development environment.

![Figure 2.1: The Modbus master sees the NET unit and the process behind it as a slave](image)

The data from the process activates a certain event channel and command procedure in the base system. This command procedure sends the information forward to the Modbus slave CPI application program and the Modbus master.

When COM 500i is used, all protocol specific objects are created automatically (for example, command procedure and event channels). This is recommended since otherwise the user has to create an application, which forwards the data between the process device and the Modbus protocol.
2.2 Related documents

The following MicroSCADA manuals should be available for reference during the use of this manual:

<table>
<thead>
<tr>
<th>Name of the manual</th>
<th>MRS number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS 600 9.3 System Configuration</td>
<td>1MRS756646</td>
</tr>
<tr>
<td>SYS 600 9.3 System Objects</td>
<td>1MRS756662</td>
</tr>
<tr>
<td>SYS 600 9.3 Application Objects</td>
<td>1MRS756660</td>
</tr>
</tbody>
</table>

2.3 Document revisions

<table>
<thead>
<tr>
<th>Version</th>
<th>Revision number</th>
<th>Date</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9.3</td>
<td>31.3.2010</td>
<td>New document</td>
</tr>
<tr>
<td>B</td>
<td>9.3 FP3</td>
<td>25.11.2013</td>
<td>Updates</td>
</tr>
</tbody>
</table>
3 Safety information

This chapter gives information about the prevention of hazards.

3.1 Backup copies

We suggest that taking backup copies before making any changes, especially ones that might have side effects. Software and data need to be copied to another place, usually to a CD or a backup tape. A writable CD and DAT tape are commonly used.

Backup copies make it easier to restore application software in case of a disk crash or any other serious failure resulting in the loss of stored data. Therefore, it is recommended that backup copies are taken regularly.

There should be at least two system backup copies and two application copies. A new backup is copied over the oldest backup. This way the latest version is always available, even if the backup procedure fails.

Detailed information on how to take backup copies should be delivered to the customer with the application.

System backup

Usually, a system backup is taken after the application is made. A backup should be taken every time changes are made to the SYS 600 system. For example, if the driver configuration or the network set-up is changed.

Application backup

An application backup is taken simultaneously with the system backup after the application is made. A backup should be taken every time changes are made to the application. For example, if pictures or databases are edited or new pictures are added.

3.2 Fatal errors

A fatal error is an error that causes a break-down or a locked situation in the SYS 600 program execution.

Handling

In case of a fatal error:

1. Write down the possible SYS 600 error messages.
2. Shut down the SYS 600 main program. If this cannot be done in the SYS 600 Control Panel, try to end the task in Windows Task Manager.
Shutting down the base system computers by switching off the power may damage files.

3. In Windows, the data kept in the main memory at the time of a fatal error is placed in the drwtsn32.log file. It is placed in a system folder, for example, WINNT. Analyze and copy the data in this file.

4. Restart the system.

Report the program break-down together with the possible SYS 600 error messages and the information from the drwtsn32.log file to the SYS 600 supplier.

**Status codes**

Error messages in SCIL are called status codes. A list of status codes and short explanations can be found in SYS 600 Status Codes.
4 Instructions

4.1 General

CPI application program

A program that has been made by using the Communication Programming Interface (CPI) software is called a CPI application program. This program is a separate executable file, Modbus_Slave.exe, which locates in the \prog\Modbus_Slave directory. A CPI application program is connected to the SYS 600 base system through the TCP-IP network. The Modbus slave CPI application program communicates with the Modbus masters via the serial ports or the TCP/IP ports of the computer running it. Modbus slave supports the use of two serial and two TCP communications, which can all be used simultaneously. The used communication channels can be defined in the config.ini configuration file. With each serial port the Modbus slave can only be connected to one Modbus master at a time. With the TCP/IP communications, multiple masters can communicate with the slave simultaneously.

Requirements

The following software is required:

- SYS 600
- Windows operation system
- Modbus slave executable
- MDDebugWin.exe (debug window executable)

The Modbus slave application can be installed either on the same computer as the SYS 600 base system or on a separate computer. The most common and recommended setup is that the modbus slave instances are connected to the same computer as the SYS 600 base system and the COM 500i application.

Installation

Modbus Slave software is installed when SYS 600 is installed. modbus_slave.exe and the configuration file config.ini are installed to the /sc/prog/modbus_slave directory. If an older config.ini file is found, it is overwritten. A config.ini file is created from a new installation file.

4.2 Configuration

General

The configuration can be divided into following parts:

- Base system configuration
- Communication system configuration
• COM 500i configuration

The base system configuration can be done by using SCIL statements. The communication system configuration is done by configuring the Modbus slave CPI application program. When COM 500i is used, the station objects configured to a Modbus slave node should be defined on the NCC page of the Signal X-reference tool.

4.2.1 Base system configuration

General

Each base system has a set of objects that specify the base system and its environment, hardware and software, as well as the physical and logical connections of the base system and its applications.

The base system objects are defined with SCIL commands in the SYS_BASCON.COM file, which is executed every time the base system is started. With a few limitations, the definition and modification of the base system objects is possible at any time when SYS 600 is running. During the operation, the base system objects are in the primary memory of the base system computer.

The Modbus slave CPI application program emulates an RTU type station, which means that the interface towards the base system is similar as with an RTU station. The CPI application program requires a node of its own. Communication between this node and the base system takes place via a LAN link.

Configuration steps

To configure SYS_BASCON.COM:

1. Define the base system.
2. Define a LAN link.
3. Define an application.
4. Define nodes for the Modbus slave instances
5. Define the RTU stations. The number of RTU stations can be the same as the number of the connected process units. This way the information related to a certain process unit can be routed to the corresponding station in the NET unit. This way, the Modbus master can differentiate between the information from different process units. In the COM 500i configuration, it is possible to use a maximum of 8 NCC stations.

The definitions are made in the example below. For more information on the system objects, see SYS 600 System Objects. Steps 1..3 are necessary, but they are configured using the standard SYS_BASCON.COM template. Steps 4..5 are necessary for the Modbus slave.
Example

The following is an example of the SYS_BASCON.COM file for communication with the Modbus slave protocol. An application CPI_TEST is defined. In this example, 1 node and 2 stations are configured.

;******************************************************************************;
; SYS_BASCON.COM
; BASE SYSTEM CONFIGURATION TEMPLATE
;******************************************************************************

#CREATE SYS:B = LIST(-
   SA = 209,-
   ND = 9,-
   DN = 3,-
   DS = "RTU",-
   FS = "NEVER")

;******************************************************************************;

#CREATE LIN:V = LIST(-
   LT = "LAN")

#CREATE LIN1:B = %LIN

;******************************************************************************;

#CREATE NOD3:B = LIST(-
   LI = 1,-
   SA = 203)

;******************************************************************************;

;******************************************************************************

;******************************************************************************;

;******************************************************************************

;******************************************************************************;

#LOOP_WITH I = 1..5
#CREATE MON’I’;B = LIST(-
   TT = "LOCAL",-
   DT = "X") ;X MONITOR
@MON_MAP(%I) = -1
#LOOP_END

#LOOP_WITH I = 6..10
#CREATE MON’I’;B = LIST(-
   TT = "LOCAL",-
   DT = "VS") ;VISUAL SCIL MONITOR
@MON_MAP(%I) = -1
#LOOP_END

;******************************************************************************;
4.2.2 Communication system configuration

General

Unlike configuring a protocol implemented in PC-NET, configuring the communication system for the Modbus slave protocol is done by configuring the CPI application program. This can be done by setting the configuration parameters of the file config.ini located in the sc\prog\modbus_slave directory. Config.ini can be edited by using a text editor, such as Notepad. The parameters should be set before the Modbus slave CPI application program instances are started.

The communication parameters of a Modbus slave CPI application program can be divided into six groups:

- Own Node Parameters
- Base System Parameters
- Serial Port Communication Parameters
- TCP/IP Communication Parameters
- Modbus Slave Parameters
- Debug Parameters
The parameters are grouped into sections in the configuration file, which are separated by section headers written inside square brackets. For example, the serial port communication parameters are listed below the section header [SerialPortCommPar]. The names of the sections and parameters should not be modified. If these names are modified, the program will use default values.

**Own node parameters**

The following parameters describe the node and communication channels of the Modbus slave application program:

- **own_node_no.** The number of the node of the CPI application program.
- **own_station_no.** The station address of the node of the CPI application program.
- **serial_port_in_use.** Sets whether or not the serial port communication 1 is used. Values: 1 – in use, 0 – not in use.
- **serial_port_2_in_use.** Sets whether or not the serial port communication 2 is used. Values: 1 – in use, 0 – not in use.
- **tcpip_1_in_use.** Sets whether or not the TCP/IP communication 1 is used. Values: 1 – in use, 0 – not in use.
- **tcpip_2_in_use.** Sets whether or not the TCP/IP communication 2 is used. Values: 1 – in use, 0 – not in use.

If all of the communication channels are configured as not to be used, Serial port 1 is opened as default.

**Base system parameters**

The following parameters describe the connection to the base system:

- **basesystem_node_no.** The number of the base system node (the ND attribute of the SYS:B object). Value range: 1...250.
- **basesystem_station_no.** The station address of the node of the base system. (the SA attribute of the SYS:B object).
- **tcp_ipadd.** The TCP/IP address of the base system computer.

If the Modbus slave and base system exist on the same computer, IP-address 127.0.0.1 should be used.

- **application_number.** The number of the SYS 600 application the CPI application program communicates with.

**Serial port communication parameters**

The following parameters describe the properties of the serial port communications. Both serial communications have the same configurations, but they can be configured separately. The configurations are separated with sections [SerialPortCommPar] and [SerialPortCommPar2]. The configurations are:
- **port_name**: Name of the Serial port, for example COM1.
- **baud_rate**: Valid values: Depends on the hardware. Normally a value of at least 9600 bits/s is supported.
- **parity**: Valid values: 0-None, 1-Odd, 2-Even, 3-Mark, 4-Space.
- **stop_bit**: The following are the valid values. If the value is:
  - 0 - 1 stop bit is used
  - 1 - 1.5 stop bits are used
  - 2 - 2 stop bits are used

  The Stop_bit parameter value is not the actual amount of stop bits.

- **data_bits**: Valid value = 8 (RTU mode of Modbus protocol is used).
- **flow_control**: Valid values: 0 - None, 1- Xon/Xoff, 2 - Hardware.
- **dtr_control**: Valid value = 0, DTR signal is not controlled, = 1, DTR signal is controlled (Default = 1, if not specified). For more information see chapter 4.5.2, section DI Database Initialized.

If there are problems with the COM port, try to set the COM port configuration of Windows to be the same as in the Initialization file.

**TCP/IP communication parameters**

The following parameters describe the properties of the TCP/IP communications. Both TCP communications have the same configurations but they can be configured separately. The configurations are separated with sections [TCPIPCommPar1] and [TCPIPCommPar2]. The configurations are:

- **slave_ip_addr**: Defines the local IP address of the Modbus slave server.
- **tcp_ip_port**: Defines the local TCP port of the Modbus slave server. Default port number is 502.

Both Modbus Slave TCP communications need to have either separate IP address or separate TCP port number. If both are configured with same IP and TCP port numbers, neither of the communications can be opened.

- **no_of_masters**: The allowed number of simultaneously connected master. Maximum number is 16.

This value determines also the number of connections from a single master. The maximum number of connections for one TCP communication is 16. Number
of connections per master is set by the Modbus slave so that each master has the same amount of connections within the 16 connections. For example if the no_of_masters is 3 the number of connections for one master is set to 5 (in this case one connection slot is not used). The recommended number of masters is 2, 4 or 8, which allows for the best use of available connections.

- no_of_ips. The number of allowed IP addresses. Maximum number of allowed IP addresses is 32.
- allowed_ip_addr_x. Allowed IP address. The "x" is the number of the IP address.

  The IP addresses must be given in the form xxx.xxx.xxx.xxx (for example, allowed_ip_addr_1 = 127.0.0.1). Only IP addresses listed in this configuration are allowed to make a connection to the Modbus server. Connections from other IP addresses are discarded.

- connection_watchdog defines the time a connection waits for a request from a master before the connection is closed as unused. The watchdog time is given in seconds and the default value is 20

**Modbus slave parameters**

The following general parameters are in the Initialization file. They have default values, which can be used in normal cases. However, these values can be edited if there is a need to create some special features.

- address_offset. Defines how the Modbus master address is mapped in the slave device.

According to the Modbus protocol standards, all the addressing is with reference to zero.

Read analog value from register 40001 (set in Modbus master) is read from the address 0000 of the slave device. In this case the value of offset is 1.

For the SYS 600 Modbus master there is no shifting of addresses as above.

Read analog value from register 40001 (set in master) is read from address 0001 of the slave device. In this case the value of offset is 0.

For more information about the usage of the Address_offset parameter, see Section 5.2.4

Addressing schematics with Modbus protocol

- change_bit_offset. This configuration indicates if change bits are used and what is the offset in memory address between a process object value and its change bit. A change bit indicates if the state of a binary process object has changed more than once between read requests.
Change bit value 0 means that the process objects state has not changed or that the changed state can be seen in the objects current value.

Value 1 means that the process objects state has changed more than once between the last polled state and the current state.

The actual process objects and their memory bits are stored in the same container, so the change bits can be read the same way, and in the same poll as the process object. A change bit is created only for binary process objects when they are written. Each binary process object in Modbus slave has only one change bit. When creating and cross-referencing binary process objects, the user should be cautious not to assign addresses that overlap with other objects change bits. When a change bit is read, the bit is cleared. If the change bit is not read at all, the value of the change bit stays unchanged. The change bit functionality can only be used with one master due to clearing of a change bit after it is read. Figure 4.1 below shows the functionality of a change bit when a process objects value changes and when a read message (dashed line) occurs.

Value: 0 – not in use,

Values: 1 ... 32767 – change bit is used, and this is also the number of offset in bits. Recommended offset values are 1 and 16, in which the change_bit is always the next bit or the same bit in the next word/byte.

If the change_bit_offset is 1, a binary process object in address: block 2 bit 0 has its change bit in address to block 2 bit 1.

If the change_bit_offset is 16, a binary process object in address: block 3 bit 1 has its change bit in address to block 4 bit 1.

Figure 4.1: Change detection bit

- modbus_master. Type of the Modbus master. If SYS 600 is used as the Modbus master, this parameter should be given value 1. If other master system than SYS 600 is used, this parameter should be given value 0.
- no_of_stns. Number of Modbus stations configured in the system. The maximum number of stations used in one Modbus slave is 8.
• stn_no_. Station numbers of the Modbus stations. Should be given as a separate parameter for each station. For example, in the case of two parameters, parameter stn_no_1 determines the station number of the first station and stn_no_2 the number of the second station. The station numbers do not need to be sequential.

• format_analog. The format of analog values. Valid values are 5, 6 and 7 as shown in Table 4.1 below.

Table 4.1: The format of analog values

<table>
<thead>
<tr>
<th>Value</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>WORD</td>
<td>Integer value, two bytes long (Unsigned if SYS 600 is used as Modbus master and Signed for any other Modbus master).</td>
</tr>
<tr>
<td>6</td>
<td>LONG</td>
<td>Signed 32 bit value, four bytes long in msb-lsb order*</td>
</tr>
<tr>
<td>7</td>
<td>MSB_LONG</td>
<td>Signed 32 bit value, four bytes long in lsb-msb order*</td>
</tr>
</tbody>
</table>

* The most significant byte - the least significant byte order (and vice versa).

• respond_also_if_invalid. This parameter defines whether or not an exception message is sent the master if the status of data in the requested station is not ok. Valid values: 0 = Modbus requests are responded to with exception 0xB if data is invalid, 1 = Modbus requests are responded to even if data is invalid (Default = 1).

• se_applevent_used. This parameter sets the Modbus slaves' SE attribute, which is related to system and node status messages and use of the Self System Supervision (SSS). If the SSS is not used, this parameter should be set to 1, which is also the default value. If the SSS is used, this parameter needs be set to value 4. In this case, the status of the node and stations are updated to the SSS process objects. Information about the SSS process objects is presented in System Self Supervision, on page 24. For more information about the SSS, see Application design manual chapter 16.

Value 1 = SSS is not used. Also the default value.

Value 4 = SSS is used.

Debug parameters

The following parameters describe the properties of the debug functionality. The debug functionality affects the whole Modbus slave regardless of the used communications.

Debug prints slow down the serial and TCP communications to some extent. It is recommended to turn off the debug prints, if the highest communication speed is desired.

• debug_info. This parameter defines if the debug messaging is activated. If debug messaging is not needed this parameter should be set to value zero.

If the user wants to see the communication handling of the Modbus slave, value 20714 must be set to this parameter. It is recommended to use the debug mode only temporarily.
• debug_window. Defines whether or not a separate debug window is used. The debug window is started from a separate executable MDDebugWin.exe. This feature should be used if debug messages are needed and the Modbus slave is started from the APL_INIT procedure, as explained in Section 4.4 Start-up. Valid values: 1 – is used, 0 – not used.

  The start-up of the separate debug window is explained at the end of chapter 4.4.

• namedpipe_name. Defines the name of the Namedpipe, which is used to communicate between the Modbus slave and the separate debug window. The name given here needs to be used as a startup parameter when starting the debug window.

  The maximum length of the namedpipe_name is 10 characters. It can consist of letters and numbers.

• error_debugs_only. Defines if only error messages are printed into the used debug window. Valid values: 1 – only error messages are printed, 0 – both error and communication related messages are printed.

Examples

This is an example of the Modbus slaves' config.ini file with the configuration parameters explained above. Modbus slave with two stations and all four communication channels are configured in this example.

[OwnNodeParameters]
own_node_no = 3
own_station_no = 203
serial_port_in_use = 1
serial_port_2_in_use = 1
tcpip_1_in_use = 1
tcpip_2_in_use = 1
[BaseSystem1]
basesystem_node_no = 9
basesystem_station_no = 209
tcp_ipaddr = 127.0.0.1
application_number = 1
[SerialPortCommPar]
port_name = COM1
baud_rate = 1200
parity = 0
stop_bit = 0
data_bits = 8
flow_control = 0
dtr_control = 1
[SerialPortCommPar2]
port_name = COM1
baud_rate = 1200
parity = 0
stop_bit = 0
data_bits = 8
flow_control = 0
dtr_control = 1
[TCPIPCommPar1]  
slave_ip_addr = 10.0.1.1
tcp_port = 502
no_of_masters = 4
no_of_ips = 3
allowed_ip_addr_1 = 10.0.12.1
allowed_ip_addr_2 = 10.0.12.2
allowed_ip_addr_3 = 10.0.12.3
connection_watchdog = 20
[TCPIPCommPar2]  
slave_ip_addr = 10.0.1.2
tcp_port = 502
no_of_masters = 4
no_of_ips = 2
allowed_ip_addr_1 = 10.0.12.1
allowed_ip_addr_2 = 10.0.10.1
connection_watchdog = 20
[ModbusSlavePar]  
address_offset = 0
change_bit_offset = 0
modbus_master = 1
no_of_stns = 2
stn_no_1 = 1
stn_no_2 = 2
format_analog = 5
respond_also_if_invalid = 0
se_applevent_used = 1
[Debug]  
debug_info = 0
ddebug_window = 0
namedpipe_name = namedpipe
error_debugs_only = 0

In order to add one more station with address 10 under the Modbus slave parameter, add the following information:

no_of_stns = 3
stn_no_1 = 1
stn_no_2 = 2
stn_no_3 = 10

4.3

After configuration

For each input signal from the process devices, the process database should contain a process object whose value changes after the process data is received. The change activates an event channel, which in turn starts a command procedure. The command procedure sends the value to the CPI application program from which the data is transferred to the Modbus master through the communication media.
In addition to the configuration of the base and communication systems, following steps are needed:

1. Configure the Modbus master in NCC.
2. Configure the base system for process communication.
3. Configure the process units.
4. Create and define input and output process objects for the process communication. This is usually done when creating the station picture by using standard functions from an application library.
5. Define the COM 500i cross-references for signals. For more information, see SYS 600 COM 500i User’s Guide.

If COM 500i is used, the cross-references between the process objects and the Modbus slave are made in the Signal X-references Tool. COM 500i creates the required event channels and command procedures automatically. For more information, see SYS 600 COM 500i User’s Guide.

If COM 500i is not used, for more information on how to program the command procedures and values of the attributes, see Chapter 5 Technical description in this document.

### 4.4 Start-up

The Modbus slave protocol runs in an external executable program and must be started separately. The basic idea of the start-up is similar to stand-alone and HSB systems. The configuration of the HSB system is described in the System Configuration manual, chapter Configuring redundancy.

As mentioned earlier in chapter 4.1, a Modbus slave can have two serial and two TCP communication channels running simultaneously. Both TCP communications can communicate with multiple masters, but serial port communications can only have a single master. If different Modbus slave IP addresses or more master connections than one slave can co-operate with are needed, separate Modbus slave instances must be configured.

This example must be applied to the required amount of Modbus slave lines, i.e. NCC connections from COM 500i. The following steps are required to configure 3 Modbus slave instances:

1. Create the following directories:

   * \sc\prog\modbus_slave\s1
   * \sc\prog\modbus_slave\s2
   * \sc\prog\modbus_slave\s3

2. Copy the modbus_slave.exe and config.ini to each of the created directories.
3. Rename the modbus_slave.exe to Mds1.exe, Mds2.exe and Mds3.exe. The renaming is needed to shut down the instances in a controlled way. If there is only one instance of the modbus slave, renaming it is not necessary and the shutting down (step 6) can be done using the name modbus_slave.exe.
4. Edit the config.ini configurations in each directory for that specific Modbus slave instance and make the corresponding configuration to the base system as instructed in Section 4.2.1 Base system configuration in this manual. The number of the main application should be given to the configuration item "application_number".

   A unique node must be created for each Modbus slave instance.

5. Create a .bat file Mds1.bat into the modbus_slave\s1 directory, with the following example content:

   cd C:\sc\prog\modbus_slave\s1
   mds1

   Create a similar .bat file for each instance in their own subdirectory. A .bat file can be created with, for example, Windows Notepad.

6. Add the following script to a command procedure executed from the event channels APL_INIT_1 and APL_INIT_H (only in HSB systems) of the main application.

   ;MD_SLAVE_START:C
   @MDS1_STATUS = OPS_CALL("C:\sc\prog\modbus_slave\s1\MDS1.BAT",0)
   @MDS2_STATUS = OPS_CALL("C:\sc\prog\modbus_slave\s2\MDS2.BAT",0)
   @MDS3_STATUS = OPS_CALL("C:\sc\prog\modbus_slave\s3\MDS3.BAT",0)

7. Add the following script in a command procedure executed from the event channel APL_CLOSE of the main application.

   ;MD_SLAVE_STOP:C
   @MDS1_STATUS = OPS_CALL("taskkill /IM mds1.exe /F",0)
   @MDS2_STATUS = OPS_CALL("taskkill /IM mds2.exe /F",0)
   @MDS3_STATUS = OPS_CALL("taskkill /IM mds3.exe /F",0)

8. Define the NCC connections to the Signal X-reference tool in the base system. The entered RTU station numbers should be equal to the values entered in config.ini files, item stn_no_*. COM 500i initializes the Modbus databases in the order NCC1, NCC2, and so on. If some of the Modbus slaves require a faster communication start-up after a takeover, they should be configured into a lower NCC number compared to the others.

   When running a Modbus slave from version SYS600 9.3FP2 and newer on a Windows XP or Windows 2003 server, it is necessary to start the Modbus slave with a special flag '-closedisable'. This flag disables the possibility to stop the Modbus slave from the upper right corner of the Modbus slave console window. In Windows 7 and on the Windows 2008 server, this flag should not be used, since the Modbus slave is usually running in session 0 (user MicroSCADA). Starting example for Windows XP or Windows 2003 server:
If the serial port or TCP port cannot be opened when the Modbus slave is started, or if the sending fails during communication, the opening is reattempted at a regular interval. This feature is useful, if the serial port is virtual or USB-based.

### Debug window start-up

The separate debug window is started from a standalone executable MDDebugWin.exe file. The debug window does not require any configurations, except the name of the namedpipe defined in the Modbus slaves’ config.ini file. The exact same name must be given as a start-up parameter to the debug window, for example, by using a .bat file.

Example of the debug window start-up command with an example pipename parameter:

```bash
MDDebugWin.exe mdncc1
```

For the debug printing to work, also the debug_window must be set to 1 and the debug_info set to 20714 in the Modbus slave config.ini file.

### System Self Supervision

The status of the Modbus slave can be supervised with the System Self Supervision (SSS). The use of the supervision functionality is controlled with the SE (System Messages Enabled) attribute, which can be set in the config.ini configuration file. The SE attribute value can be read in the Test Dialog tool with the SCIL command NETx:SSE, where the "x" is the number of the Modbus slave node. If the value of the SE attribute is not defined, the default value 1 is used.

With the setting SE = 1, the SSS is not used and the system messages are not enabled. The functionality is the same as in SYS600 9.3 FP2. A status update in the process object type ANSI Analog Input (UN=0, OA=8000+STA number) can be used, but this feature is not supported in the current SSS version. The status of a station can also be checked by reading the related Object Status (STAx:SOS) attribute.

With the setting SE = 4, the SSS is used and the system messages are enabled. The supervision information consists of the connection statuses of stations and nodes, which are updated in specific process objects by the SSS. The System Configurations Tool does not have support for the Modbus slave and the SSS stations must be created manually. The station process objects are type ANSI Analog Input (UN=0, OA=1000+STA number) and type ANSI Binary Input (UN=0, OA=1000+1000000hex+STA number). The node process objects are type ANSI Analog...
Input (UN=0, OA=6000+Node number) and type ANSI Binary Input (UN=0, OA=6000+1000000hex+Node number). In the SE = 4 mode, the Modbus slaves' interface with the SSS is similar to External OPC DA Client with IEC 61850, setting SE = 4.

4.5 How to test the configuration

SYS 600 as the Modbus master

The configuration can be tested before the actual Modbus master is connected to the system by using SYS 600 as the Modbus master. By using this configuration, it is possible to verify if the communication between the Modbus slave and master is working properly.

The Modbus master protocol is implemented in the PC-NET software. The Modbus master can be configured on the same computer as the Modbus slave or on a separate computer. It is recommended to have the Modbus master on a separate computer.

To configure the base system as the Modbus master, the following base system configuration should be done:

1. Define the base system, monitors and application, if they are not the same as those defined for the Modbus slave.
2. Define a node and a link for the NET unit.
3. Define a PLC station for each station configured in the Modbus slave.

   This step is not needed if System Configuration Tool is used.

The following communication system configuration is required:

1. Define a Modbus master line.
2. Define the PLC stations with the same logical address as the RTU stations.

The station addresses (SA attribute) of the PLC stations should be equal to the stn_no_* parameters of the Modbus slave stations. Other attributes of the Modbus master line and PLC stations should also match the configuration of the Modbus slave. For more information on the attributes, see SYS 600 System Objects.

Below is an example for defining the station type and two PLC stations.

;Define PLC stations. These setting are needed only if System Configuration Tool is not used
#CREATE STA1:B = LIST(-
   TT = "EXTERNAL",-
   ST = "PLC",-
   ND = 1,-
   TN = 1)  
#CREATE STA2:B = LIST(-
   TT = "EXTERNAL",-
   ST = "PLC",-
   ND = 1,-
5 Technical description

5.1 Modbus protocol

The Modbus Communications Protocol is an asynchronous, byte packaged protocol used for communication between the master stations and Intelligent Electronic Devices (IEDs) or Remote Terminal Units (RTUs). It supports serial and TCP/IP communications and provides a transport mechanism for the master’s requests and RTU response messages. It supports up to 247 RTUs on a multi-drop line.

The Modbus slave application can have two serial and two TCP/IP communications running simultaneously and it supports one master with each serial line communication and up to 16 masters with each TCP/IP communication.

The Modbus protocol has two distinct modes: ASCII Modbus, which uses ASCII-encoded hexadecimal messages and binary Modbus, which uses raw binary messages. The Modbus slave implementation described in this document only supports the binary mode.

All transactions are initiated by transmission of a request from the master station. An RTU may not transmit unsolicited information. Every master station request must be addressed to a specific RTU, and some implementations of Modbus do not support the broadcast message request type. A transaction consists of a single master station request followed by an RTU response, an exception frame or a master station timeout, if no RTU response is generated.

5.2 Communication

5.2.1 CPI application program

The Modbus slave protocol is implemented in SYS 600 by using Communication Programming Interface (CPI) software. CPI is a collection of functions written in C language suitable for implementing protocol converters between the internal protocol of SYS 600 and other protocols. The internal protocol of SYS 600 is used in communication between the SYS 600 nodes, for example, between a base system and a NET unit. The communication between the CPI application program and the SYS 600 base system is based on the TCP/IP network.

5.2.2 Connection to a SYS 600 base system

The CPI application program establishes a connection to the base system with the TCP/IP address as specified in the Modbus slave’s configuration file. After establishing the connection, a link is established to the application specified in the configuration file. The CPI application program can access the SYS 600 process database using the CPI library functions.
SYS 600 can have a number of slaves connected to it. The CPI application program acts as a communication front-end and is seen as a node in the SYS 600 system. The stations configured in SYS 600 can also be logically created in the CPI application program. For Modbus slave emulation, SYS 600 is seen as a data collector.

### 5.2.3 Data flow

Figure 5.1 describes the data flow between the process devices and the Modbus master. Both directions are described separately since the data is handled in a different way depending on the direction.

See COM 500i User's guide for detailed information about the data flow configuration.

**Figure 5.1: Data flow between the process devices and the Modbus master**

#### Input data

When input data, for example, indications and measured values, are sent from the process devices to the Modbus master, the following steps are taken:

1. The process devices send the data to the SYS 600 process database.
2. The updated process object activates an event channel.
3. The event channel executes a command procedure. Some of the attributes of the process object are given as arguments to the command procedure.
4. The command procedure sends the data to the database of the CPI application program based on specific cross-reference information.
5. The CPI application program sends the data to the Modbus master.

The cross-reference information is needed to deliver data to the database of the CPI application program, for example, object address and message type. The number of event channels and command procedures needed to deliver data to the master depends on the application. One solution is to have one command procedure for each process object type. Examples of the command procedures are given later in this document. If COM 500i is used, all the required command procedures and event channels are created automatically.

**Output data**

When output data, for example, object commands and analog setpoints, are sent from the Modbus master to the process devices, the following steps are taken:

1. The command is received by the SYS 600 process database. There must be a separate input process object for each Modbus command address. These process objects are created automatically when COM 500i is used.
2. The updated process object activates an event channel.
3. The event channel executes a command procedure. Some of the attributes of the process object are given as arguments to the command procedure.
4. The command procedure sends commands to the process devices by setting the corresponding output process object(s) and, if required, sends a confirmation to the Modbus master via the CPI application program.

Cross-reference data can also be used with commands. It can contain, for example, information on the logical names and indices of the output process objects. Examples of the command procedures are given later in this document.

**5.2.4 Addressing schematics with Modbus protocol**

Figure 5.2 shows the addressing schema with analog values in the Modbus protocol when using the Modbus slave in SYS 600 for example with the COM 500i application and a Modbus type of NCC. Analog values are mapped to address space 4xxxx in the protocol definition. The memory area is not used in protocol queries and responses, and it can be defined from the function type. Protocol addressing is defined in following way:

For example, register 40001 is in query address 0000 and one register is built up of 16 bits (word). If the 3rd party master uses an address offset, the offset parameter must be set to 1, so that the wanted register number and the used block number in the SYS 600 application match each other. If the offset parameter is 0 in this case, it must be remembered that the used address is different in the master and the slave.

If SYS 600 is also the master system, the offset value should be set to 0. The SYS 600 master queries do not use address offsets. The defined register in the master configuration is shown in protocol with the same address number. The SYS 600 configuration does
not allow register address 0. Therefore, possible block numbers are 1...65535 on the slave's side.

Offset parameter is also used with binary data. In SYS 600 the discrete inputs and coils of the Modbus slave are mapped to RTU type blocks. Possible block address values are 1...4096 and bit values 0...15. The Modbus protocol does not have a block definition for binary data, it only uses data grouping of 8-bit bytes in query responses. The used memory area in protocol definition is 0xxxx. Coils and inputs are addressed starting from 1. However, in protocol queries the addressing begins from bit 0.

If SYS 600 is used as a master, the bit 0 in query and master database configuration means the block 1, bit 0 address in COM 500i. In the case of a 3rd party master, the bit address is 1 in the slave, if the offset parameter is used. This schema is used, if the master can define only binary addresses starting from 00001.

It must be remembered that the defined topic base address in the SYS 600 master affects the used bit address in the process object. For example, if the used base address is 1 and bit 4 is wanted from the slave database, the BI process object's bit address is 4 (block 1). However, if the base address is 2 and the wanted input bit is 4, the used bit address must be 3. In this case block 1 begins from bit 2 instead of 1.
5.2.5 Device communication attributes

The following attributes are used to exchange data between SYS 600 and the Modbus master. The attributes are similar to those of the RP 570 (SPI) stations with a few exceptions.

- By default, the CPI application program does not respond, if some or all of the data requested block has an object status other than zero. This can be changed with the `respond_also_if_invalid` configuration, which is explained in Section 4.2.2 Communication system configuration. This applies to analog input, as well as to single and double indications, i.e. to function codes 1, 2, 3 and 4. If the station's In Use attribute is set to zero, all the statuses are 10. If the In Use attribute is set to 1, all the statuses are set to 2 to indicate suspicious values.

### AV

**Analog Value**

Changes in analog measured values that are to be sent to the Modbus master are written to this attribute. A vector is assigned to the AV attribute consisting of a time stamp and analog value.

**Data type:** Vector

**Value:** A vector of three elements: (TIME, VALUE, STATUS)

- **TIME**: Time stamp for the process object whose value is sent. Time as RTU_ATIME format.
- **VALUE**: With configuration 'analog_format' value 5: Analog value (scaled to -32768...32767)
  - With configuration 'analog_format' values 6 and 7: Analog value (scaled to -2147483648...2147483647)
- **STATUS**: Status code of the point. Value range: integer 0 ... 3, valid = 0

**Index range:** 1...65535

**Access:** Write only

```plaintext
;set AV block nr. 100 to 1234, status = OK (= 0 )
#SET STA99:SAV(100) = (RTU_ATIME(%RT,%RM),1234,0)
```

The time stamp information is ignored in Modbus.
**InDica­tions**

Indica­tion changes that are to be sent to the Modbus master are written to this attribute.

**Data type:** Vector

**Value:** A vector with 4 … 6 elements as follows:

(TIME, BIT_NR, BIT_VAL, STATUS [,ERMI_ENABLED [,TIME_QUALITY]])

- **TIME** Time stamp for the process object whose value is sent
  Time is given in RTU_ATIME format
- **BIT_NR** The bit number for the changed bit (0…15)
- **BIT_Val** 0…1
- **STATUS** Status code of the point. Value range: integer 0 … 3, valid = 0

**Index:** 1…4096

**Access:** Write only

The ERMI_ENABLED and TIME_QUALITY attributes are not used at the moment in the Modbus slave.

The Modbus protocol does not contain a field to indicate the validity of the data. As default, the Modbus slave does not respond to the requests initiated from Modbus master, if some or all of the data of the requested block has a STATUS other than zero. This applies to analog inputs and pulse counters as well as to single and double indications, i.e. to function codes 1, 2, 3 and 4. If the control flag 'respond_also_if_invalid' in the configuration file equals to 1, the rule above is ignored and the requests are responded to also when some of the data has an invalid STATUS.
The IU attribute indicates whether the station is in use or not. If the station has been taken out of use, no responses are sent to the serial line or to the TCP connections. Taking the station in use and out of use does not affect the validity of the data. Data updates using attributes ID, DD, AV and PC are accepted even if the station objects are out of use. If more than one STA object has been configured to the same modbus slave instance, it is recommended to have the same IU value for all of them.

When the IU is set to 0, and if the se_applevent_used configuration is 4 in the config.ini file, the station status code is set to 13489 "SPIP_STATION_STOPPED".

**IU**

**In Use**

The IU attribute indicates whether the station is in use or not. If the station has been taken out of use, no responses are sent to the serial line or to the TCP connections. Taking the station in use and out of use does not affect the validity of the data. Data updates using attributes ID, DD, AV and PC are accepted even if the station objects are out of use. If more than one STA object has been configured to the same modbus slave instance, it is recommended to have the same IU value for all of them.

When the IU is set to 0, and if the se_applevent_used configuration is 4 in the config.ini file, the station status code is set to 13489 "SPIP_STATION_STOPPED".

Data type: Integer
Value: 0 = not in use
       1 = in use
Default value: 1
Access: No limitations

**DI**

**Database Initialized**

The CPI application program sets DI to 0 at the start-up. When SYS 600 has updated all the values in the CPI database, it sets this attribute to 1. The CPI application program does not process any requests from the Modbus master before DI equals 1. In case DTR bit controlling is enabled (see control flag 'dtr_control' in Section 4.2.2 Communication system configuration ), the DTR signal of the serial port set to non-signaled state when DI = 1 and to signaled state when DI = 0. This signal can be wired to control the fallback switches if needed.

Data type: Integer
Value: 0 or 1
Access: No limitations

**OS**

**Object Status**

The current status of the Modbus slave station. If the SE attribute is configured as 4, the stations statuses are also updated to the SSS process objects and have different status codes than when the SE is configured to value 1.

Data type: Integer
Value: 13484 = Not connected to the master / connection to master lost
       13485 = Connected / Reconnected to master
If SE is 1:
       0 = STATUS_OK
       13484 = Not connected to the master / connection to master lost
       13489 = Connection stopped (IU attribute is set to 0)

Access: Read only
Analog Initialization

Writing to this attribute will initialize the analog and pulse counter values of the whole database. Writing to this attribute is allowed only when the attribute DI Database initialized is 0. When written successfully, the given VALUE and STATUS are applied to all AV and PC indices up to 65535 (Modbus addresses up to 65535).

Data type: Vector

Value: A vector of three elements:

- **TIME**: Current time in RTU_ATIME format.
- **VALUE**: With configuration 'analog_format' value 5:
  - Analog value (scaled to -32768…32767)
- **VALUE**: With configuration 'analog_format' values 6 and 7:
  - Analog value (scaled to -2147483648…2147483647)
- **STATUS**: Status code of the point. Value range: integer 0 … 3, valid = 0

Index range: No index

Access: Write only (write allowed only when DI=0)

Indication Initialization

Writing to this attribute initializes the single and double indication values of the whole database. Writing to this attribute is allowed only when the attribute DI Database initialized is 0. When written successfully, the given VALUE and STATUS are applied to all ID and DD indices up to 4096 (Modbus addresses up to 65535).

Data type: Vector

Value: A vector of three elements:

- **TIME**: Current time in RTU_ATIME format.
- **VALUE**: 0…1
- **STATUS**: Status code of the point. Value range: integer 0 … 3, valid = 0

Index range: No index

Access: Write only (write allowed only when DI=0)
PC  Pulse Counter
Pulse counter values that are to be sent to Modbus master are written to this attribute. The PC attribute is assigned a vector consisting of a time stamp, an end of period flag and a pulse counter value. The pulse counter values be stored as analog values in the CPI database. There should not be any address clash between the analog and pulse counter values.

Data type: Vector
Value: Vector (TIME, VALUE, STATUS)
Parameters are as in the AV attribute.

Index range: 1...65535
Access: Write only

DD  Double Indication
Double indications are sent to the CPI database using this attribute.

Data type: Vector
Value: (TIME, BIT_NR, BIT_VAL, STATUS [,ERMI_ENABLED[,TIME_QUALITY]])
BIT_NR is 0, 2, 4, 6, 8, 10, 12 or 14
BIT_VAL is 0, 1, 2 or 3
Other parameters as in the ID attribute

Index: 1…4096
Access: Write only

The ERMI_ENABLED and TIME_QUALITY attributes are not used at the moment in the Modbus slave.

5.3  Command procedures

5.3.1  Command procedures in COM 500i
Signals are sent from the process units to the Modbus master and commands are sent from the Modbus master to the process units. COM 500i reroutes the signals using command procedures and cross-references (see Figure 5.3).
Figure 5.3: COM 500i reroutes the signals

If Modbus slave protocol is used with COM 500i, the command procedures available for COM 500i are used. The cross-reference information is entered in the Signal Cross-Reference Tool. For more information, see SYS 600 COM 500i User's Guide. Table 5.1 shows the COM 500i command procedures and event channels used.

**Table 5.1: Used event channels and command procedures**

<table>
<thead>
<tr>
<th>Process Object Type</th>
<th>Event Channel</th>
<th>Command Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input</td>
<td>COM_USAI</td>
<td>COM_USAI</td>
</tr>
<tr>
<td>Single Indication</td>
<td>COM_USDI</td>
<td>COM_USDI</td>
</tr>
<tr>
<td>Double Indication</td>
<td>COM_USDB</td>
<td>COM_USDB</td>
</tr>
<tr>
<td>Pulse Counter</td>
<td>COM_USPC</td>
<td>COM_USPC</td>
</tr>
<tr>
<td>Digital Commands</td>
<td>COM_DSOB</td>
<td>COM_DSOB</td>
</tr>
<tr>
<td>Analog Commands</td>
<td>COM_DSAO</td>
<td>COM_DSAO</td>
</tr>
</tbody>
</table>
5.3.2 Command procedures in SYS 600

5.3.2.1 Command procedures for process data

If COM500i is not used, all read command procedures and event channels must be done by the user. The following chapter describes examples of command procedures.

The attribute interface of the Modbus slave CPI application program is similar to that of the RP 570 slave (SPI) station type in SYS 600. Details of the device communication attributes used by the Modbus slave can be found in chapter 'Device Communication Attributes'.

The connection between the SYS 600 process objects and messages to and from the Modbus master is made by using cross-reference data. Cross-reference data can be written to the FX (Free teXt) and FI (Free Integer) attributes of the process objects by using the Process Object Definition Tool.

If COM 500i is used, the upstream cross-references for indications are stored in Free-type process objects.

Cross-references for data transfer from SYS 600 to the Modbus master are written in the FX attribute, which is a string of max 30 characters. The general syntax for a Modbus cross-reference is (an example of a syntax):

\[
<\text{STA\_NUMBER.}> <\text{BLOCK\_NUMBER.}>[<\text{BIT\_NUMBER.}>]
\]

- **STA\_NUMBER**: Logical STA number (range 1...255). The value is stored as 3 ASCII digits with leading spaces added if necessary. FX string positions 1...3 are used for this value.
- **BLOCK\_NUMBER**: The block number of the Modbus master (range 1...125). Stored as 3 ASCII digits with leading spaces added if necessary. FX string positions 4...6 are used for this value.
- **BIT\_NUMBER**: Bit number (range 0...15) is the bit position in the indication data word. It is stored as ASCII digits with a leading space added if necessary. FX string positions 7...8 are used for this value. The bit number is defined only if the data point is to be sent to the Modbus master as an indication. If it is sent as an analog value, it is undefined (but reserved).

The FI attribute is used for storing the object index. Modbus commands activate digital and analog process input objects in SYS 600. The "real" output process object has the same logical name as this input (i.e. it belongs to the same group), but a different index. This index is kept in the FI attribute of the input object.
Analog inputs

Analog values are sent to the Modbus master as integers in the range \(-32768\ldots 32767\). All analog signals must be scaled to this range (or part of it).

Analog values are sent to the Modbus master by using the AV attribute. The value set to the AV attribute is:

\[(\text{TIME}, \text{VALUE}, \text{STATUS})\]

The block number of the analog input is set to the index of the AV attribute.

Time and status information is not supported in the Modbus protocol and therefore it is not sent to the master. Status information is stored in the CPI application program and the analog value is sent to the master, only if the status is OK.

All the analog values are connected to the same event channel/command procedure combination. The command procedure is activated each time the process object is updated. It reads the updated value, scales it and sends it to the CPI application program. An example of the command procedure is listed below:

```
;read cross-reference data
@STA_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),1,3))
@BLK_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),4,3))
@VAL = 'LN':POV('IX')
@T = RTU_ATIME(%RT,%RM)

;send value to CPI application
#SET STA'STA_NR':SAV(%BLK_NR) = (%T,ROUND(%VAL),0)
```

Single indications

All the binary inputs are connected to an event channel/command procedure combination. The command procedure is activated each time the process object is updated. It reads the updated value and sends it to the CPI application program.

Single indications are sent to the Modbus master by using the ID attribute. The value set to the ID attribute is:

\[(\text{TIME}, \text{BIT_NUMBER}, \text{BIT_VALUE}, \text{STATUS})\]

The block number of the single indication is set to the index of the ID attribute.

An example of a command procedure handling single indications is listed below. Note that zero status is assumed.

```
;read cross-reference data
@STA_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),1,3))
@BLK_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),4,3))
@BIT_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),7,2))
@T = RTU_ATIME(%RT,%RM)

;send value to CPI application, assume ok status
#SET STA'STA_NR':SID(%BLK_NR)=(%T,%BIT_NR,%BI,0)
```
It is not possible to send time stamp and status information in Modbus messages. See attribute description for more information.

Double indications

Double indications are not supported directly in the Modbus protocol. Double indications are converted into two single indication messages by the CPI application program.

Double indications are sent to the Modbus master by using the DD attribute. The value set to the DD attribute is:

\[(TIME, BIT_NUMBER, BIT_VALUE, STATUS)\]

The block number of a single indication is set to the index of the DD attribute.

```plaintext
;read cross-reference data
@STA_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),1,3))
@BLK_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),4,3))
@BIT_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),7,2))

;send value to CPI application
@T = RTU_ATIME(%RT,%RM)
#SET STA' STA_NR':DD(%BLK_NR) = (%T,%BIT_NR,%DB,0)
```

It is not possible to send time stamp and status information in Modbus messages. See attribute description for more information.

Pulse counters

The pulse counter values are treated as analog values by the Modbus slave. The pulse counter values are sent to the Modbus master using the PC attribute. The value set to the PC attribute is:

\[(TIME, VALUE, STATUS)\]

The block numbers of analog and pulse counter objects should be different since they are both treated as analog values.

```plaintext
;read cross-reference data
@STA_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),1,3))
@BLK_NR = DEC_SCAN(SUBSTR('LN':PFX('IX'),4,3))

;send value to CPI application
@T = RTU_ATIME(%RT,%RM)
#SET STA' STA_NR':SPC(%BLK_NR) = (%T,%PC,0)
```
5.3.2.2 Command procedures for commands

No configuration actions described in this chapter are needed if COM 500i is used.

Since the SYS 600 application sees the CPI program as an RTU type station, data that is transferred from the master to the slave must be handled as input data. When this kind of input is updated, the value is read by a command procedure and, if necessary, converted before it is written to the actual output object. The command procedure is activated through an event channel bound to the input object.

It is also possible to utilize input data to perform arbitrary internal operations in the application program.

The procedures that are presented below cover the basic cases, when SINCDAC commands and setpoints are mapped directly to the corresponding output objects. If a more complex relationship between the input and output objects is needed, application specific command procedures need to be built.

Process commands

Process commands from the Modbus master are handled as digital inputs with object addresses (OA attribute) in the range 0…2000 (object commands). When the CPI application program receives a process command message, it reads the object number (register address) from the message. It also activates a SYS 600 process object with the corresponding object address and sets its value according to the command (1 = ON, 0 = OFF).

The command is transferred to a binary output process object with a command procedure. The cross-reference between the input and output objects is accomplished by using the same logical name, but different indexes for the input and output objects. Input objects hold the index of the corresponding output object in their FI attribute. This makes it possible to use a common command procedure for most process commands. Only the commands that require validity checking or other special processing need individual SCIL procedures.

Below is a command procedure that is used when no special processing is needed.

```plaintext
;read cross-reference data
@OBJ_IX = 'LN':PFI(%IX)

;set value to output object
#SET 'LN':PBO(%OBJ_IX) = %DI
```

Analog setpoints

Analog commands from the Modbus master can be received by analog input process objects. The object address range starts from 3000 (decimal). The block number (register address) is added to get the address that is used for a particular setpoint.

Analog setpoints are received as integers in the range -32768…32767 and may have to be scaled.
An example of a command procedure for handling analog setpoints:

; read cross-reference data
@OBJ_IX = 'LN':PFI(%IX)

; scale value
@VAL = %AI-'SC':XSC(1)
@VAL = @VAL / ('SC':XSC(2)-'SC':XSC(1))
@VAL = @VAL * ('SC':XSC(4)-'SC':XSC(3))
@VAL = @VAL + 'SC':XSC(3)

; set value to output object
#SET 'LN':PAO(%OBJ_IX) = @VAL

5.4 Function codes

The used CPI application program is for Modbus slave protocol in SYS 600. The Modicon Modbus protocol has many function codes. There are different types of Modicon controllers available which supports different types of functions. There are 24 functions supported in Modbus protocol. All the controllers do not necessarily support all the functions. The Modbus function codes implemented in the Modbus CPI application program are 1, 2, 3, 4, 5, 6, 8, 11, 15 and 16. An exception response is sent to all other Modbus function codes. For details of exception responses, see Modbus Modicon Protocol Reference Guide.

The function codes implemented in this version of software are 1, 2, 3, 4, 5, 6, 8, 11, 15 and 16. These function codes can access the CPI database, which reflects the data in the SYS 600 process database.

According to the Modbus protocol, all the data is stored in registers or coils. Each register consists of 2 bytes and each coil consists of one ON/OFF bit.

In the Modbus slave CPI application with serial port communication, every message from master to slave or vice versa, has CRC (Cyclic Redundancy Check) error check bytes in the end of the message. In the Modbus TCP communication, no CRC bytes are added to the responses or to the requests. Sending a message without CRC with serial communication causes an exception.

There are four different addressable memory areas in the Modbus protocol as shown in Table 5.2.

Table 5.2: Modbus memory areas

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001...09999</td>
<td>Discrete Outputs</td>
</tr>
<tr>
<td>10001...19999</td>
<td>Discrete Inputs</td>
</tr>
<tr>
<td>30001...39999</td>
<td>Input Registers</td>
</tr>
<tr>
<td>40001...49999</td>
<td>Holding Registers</td>
</tr>
</tbody>
</table>

These memory areas can be accessed by using different function codes as shown in Table 5.3.
Table 5.3: Function codes and the corresponding memory areas

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Description</th>
<th>Memory area</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Read coil status</td>
<td>00001…09999</td>
</tr>
<tr>
<td>02</td>
<td>Read input status</td>
<td>10001…19999</td>
</tr>
<tr>
<td>03</td>
<td>Read holding register</td>
<td>40001…49999</td>
</tr>
<tr>
<td>04</td>
<td>Read input registers</td>
<td>30001…39999</td>
</tr>
<tr>
<td>05</td>
<td>Force single coil</td>
<td>00001…09999</td>
</tr>
<tr>
<td>06</td>
<td>Write single register</td>
<td>40001…49999</td>
</tr>
<tr>
<td>15</td>
<td>Force multiple coils</td>
<td>00001…09999</td>
</tr>
<tr>
<td>16</td>
<td>Write multiple registers</td>
<td>40001…49999</td>
</tr>
</tbody>
</table>

In the Modbus CPI application program there is no difference in the handling of function codes 1 and 2, as well as function codes 3 and 4. Function codes 1 and 2 access the same data. This is also the case with function codes 3 and 4.

Table 5.4 lists the Modbus function codes and the used process object types.

Table 5.4: Function codes and the corresponding process object types

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Description</th>
<th>Process Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Read coil status</td>
<td>Indication (single &amp; double)</td>
</tr>
<tr>
<td>02</td>
<td>Read input status</td>
<td>Indication (single &amp; double)</td>
</tr>
<tr>
<td>03</td>
<td>Read holding register</td>
<td>Analog Input</td>
</tr>
<tr>
<td>04</td>
<td>Read input registers</td>
<td>Analog Input</td>
</tr>
<tr>
<td>05</td>
<td>Force single coil</td>
<td>Digital input</td>
</tr>
<tr>
<td>06</td>
<td>Write single register</td>
<td>Analog Input</td>
</tr>
<tr>
<td>15</td>
<td>Force multiple coils</td>
<td>Digital Input</td>
</tr>
<tr>
<td>16</td>
<td>Write multiple registers</td>
<td>Analog Input</td>
</tr>
</tbody>
</table>

Table 5.5 shows the subfunction codes of function code 8. This function does not affect the CPI database, which also means that the process database is not affected. Some subfunction codes update the diagnostic counters in the CPI database.

Table 5.5: Subfunction codes of function code 8

<table>
<thead>
<tr>
<th>Subfunction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Return query data</td>
</tr>
<tr>
<td>01</td>
<td>Restart communication option</td>
</tr>
<tr>
<td>10</td>
<td>Clear counters and diagnostic register</td>
</tr>
<tr>
<td>11</td>
<td>Return bus message count</td>
</tr>
<tr>
<td>12</td>
<td>Return bus communication error count</td>
</tr>
<tr>
<td>13</td>
<td>Return exception count</td>
</tr>
<tr>
<td>15</td>
<td>Return slave no response count</td>
</tr>
</tbody>
</table>
The broadcast mode is not supported by any of the function codes.

5.4.1 Function codes for process data

5.4.1.1 Function code 01 - Read coil status

This function reads the value (ON/OFF) of coils from the specified address. The function can access single and double indication process objects via the CPI database. Modbus protocol does not allow direct transmission of a double indication. A double indication is transmitted as two single indications to the Modbus master. This has to be taken care of at the master’s side. The least significant bit of a double indication is transmitted first and the most significant bit last.

The query message contains the following information:

- Slave address
- Function code
- Start coil address high
- Start coil address low
- Number of coils high
- Number of coils low
- CRC - error check (Serial port communication only)

Slave address is the logical station address from which the data is requested. The coil address depends on the process object address.

The coil address for a process object is calculated in the following way:

\[ \text{Coil address} = 16 \times (\text{block number} - 1) + \text{bit no} + 1 \]

The block number for a coil address is calculated in the following way:

\[ \text{Block number} = (\text{Coil address} - 1 - \text{bit no}) / 16 + 1 \]

Coil address for a process object with block number 2 and bit number 0:

\[ \text{Coil address} = 16 \times (\text{block 2 number} - 1) + \text{bit no} 0 + 1 = 17 \]

For a double indication with block number 3 and bit number 2:

\[ \text{Coil address1} = 16 \times (3-1) + 2 +1 = 35 \]
\[ \text{Coil address2} = 16 \times (3-1) + (2 +1) + 1 = 36 \]

The address range is 1…65535. If some or all of the requested data is outside this address range, an exception response is sent indicating an illegal address value.

The response message contains the following information:

- Slave address
The number of coil values that can be read at a time depends on the master. The slave has no limitations in this regard.

5.4.1.2 **Function code 02 - Read input status**

This function is equivalent to function 01 described above.

5.4.1.3 **Function code 03 - Read holding registers**

This function reads analog values from the holding registers. The function can access analog input process objects from the process database. Different types of formats are available for reading the data as shown in Table 5.6.

**Table 5.6: Formats of analog data**

<table>
<thead>
<tr>
<th>Format Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>5</td>
<td>Integer value, two bytes long (unsigned for the SYS 600 Modbus master and signed for any other Modbus master).</td>
</tr>
<tr>
<td>WORD</td>
<td>6</td>
<td>Signed 32 bit value, four bytes long in msb-lsb order*</td>
</tr>
<tr>
<td>MSB_LONG</td>
<td>7</td>
<td>Signed 32 bit value, four bytes long in lsb-msb order*</td>
</tr>
</tbody>
</table>

* The most significant byte - the least significant byte order (and vice versa).

The format of analog data is defined in Modbus slave with the format_analog parameter in the config.ini. This parameter affects all analog values.

The format configured in the configuration is used. If format 5 is used, each piece of data is two bytes long. The format has to be chosen based on the master. All the above formats are implemented in the SYS 600 master.

If format 6 or 7 is used, it must be remembered that a single value takes two blocks from the address base.

The query message contains the following information:

- Slave address
- Function code
- Start register address high
- Start register address low
- Number of registers high
- Number of registers low
- CRC - error check (Serial port communication only)
Slave address is the logical station address from which the data is requested. The start register address is the same as the block number of the first process object. For example, if the block number of a process object is 10, the register address is also 10.

The address range is 1…65535. Any request for data outside this range results in an exception response.

The number of registers that can be requested in one message is limited to 125. If more than 125 registers are requested, an exception response is returned.

5.4.1.4 Function code 04 - Read input registers

This function code is equivalent to function code 03 described above.

5.4.2 Function codes for commands

5.4.2.1 Function code 05 - Forcing a coil

This function forces the value of a coil to be either 0 or 1. This function sends an object command in case SYS 600 is the Modbus master. When this message is received from the master, the slave treats it as a digital value and forces this process object to 1 or 0. The command procedure in the SYS 600 Modbus slave forwards these commands to the process side. For more information about this subject, see the Process commands in Section 5.3.2.2 Command procedures for commands.

The query message contains the following information:

- Slave address
- Function code
- Coil address high
- Coil address low
- Forcing value high (00 or FF hexadecimal)
- Forcing Value low (00 hexadecimal)
- CRC - error check (Serial port communication only)

The slave address is the logical address of the station to which the command is sent. The coil address is the object number of the process object. For example, if the block number of an object command is 100, the coil address is also 100. The normal response message to this query is an echo of the query itself.

5.4.2.2 Function code 06 - Modify register content

This function code sets a single register to a particular value specified in the query message. In the case of the SYS 600 Modbus master, this is an analog set point. The command procedures forward these commands to the process side. For more information
about this subject, see Analog setpoints in Section 5.2.4 Addressing schematics with Modbus protocol.

On the slave's side this command modifies the corresponding analog input process object with the object address 3000 analog offset + register address.

The query message contains the following information:

- Slave address
- Function code
- Register address high
- Register address low
- Data high
- Data low
- CRC - error check (Serial port communication only)

The slave address is the logical address of a station to which this command is sent. The register address is same as the object address of the process object. For example, if the object number is 200, the register address is also 200. The valid address range is 1…1095. Any register address outside this range results in exception response.

Only two-byte signed integer values can be sent to the slave. If floating point values are sent, the decimal portion is truncated.

### 5.4.2.3 Function code 08 - Diagnostics

This function serves to check the communication between the slave and the master. The following sections describe the subfunction codes of the function code 8.

#### Subfunction code 00 - Return query data

The data passed in the query data field is to be returned in the response. This is just for checking the line between the slave and the master.

Both the query and response messages contain the following information:

- Slave address
- Function code
- Subfunction code high
- Subfunction code low
- Data field high
- Data field low
- CRC - error check (Serial port communication only)

#### Subfunction code 01 - Restart communication option

This function initializes the database and clears the diagnostic counters. If the slave is in the Listen Only mode, no response is sent to the master but the slave is forced out of
the Listen Only mode. If the Listen Only mode is not used, a normal response is sent. The query structure is the same as above.

This message sent to any of the slaves in SYS 600 forces all the stations connected to it out of the Listen Only mode. However, restart is considered as a station-specific message.

**Subfunction code 04 - Force Listen Only mode**

Forces the slave into the Listen Only mode. In the Listen Only mode, the slave receives messages normally, but none of the messages are handled, except for the function code 8 subfunction code 1 message, which turns the Listen Only mode off. The Listen Only mode is used for monitoring communication.

No response is sent to this message.

**Subfunction code 10 - Clear counters and diagnostic register**

The diagnostic register is not relevant to SYS 600. This function clears all the counters. Counters are also cleared on power up. There is only one set of diagnostic counters for the SYS 600 Modbus slave. There are no separate counters for individual stations connected to SYS 600. This applies wherever counters are mentioned. The query and response are the same as above.

**Subfunction code 11 - Return bus message count**

The response data field contains the quantity of messages that the slave has detected on the communication system since power up as follows:

- Slave address
- Function code
- Subfunction code high
- Subfunction code low
- Data field high
- Data field low
- CRC - error check (Serial port communication only)

**Subfunction code 12 - Return bus communication error count**

The response data field returns the number of CRC errors encountered by the slave since its last restart or power up.

The query and return messages contain the following information:

- Slave number
- Function code
- Subfunction code high
- Subfunction code low
- Data field high
- Data field low
• CRC - error check (Serial port communication only)

**Subfunction code 13 - Return bus communication error count**

The response data field returns the quantity of Modbus exception responses. The query and return fields are the same as in code 12.

**Subfunction code 14 - Return slave message count**

The response data field returns the quantity of messages addressed to the slave which the slave has processed since start-up. The query and response fields are the same as in code 12.

**Subfunction code 15 - Return slave no response count**

The response data field returns the quantity of messages addressed to the slave to which no response has been returned. The query and response fields are the same as in code 12.

For subfunction codes 03, 16, 17, 18, 19 and 20 the data fields of the request are returned in the response, because these function codes are either hardware dependent or not relevant.

The subfunction code 21 is not supported.

**5.4.2.4 Function code 11 - Fetch communication event counter**

This function code returns a status word and an event count (2 bytes each). The event count is incremented for each successful message completion. This counter is not incremented for exception responses or fetch event counter commands. The events counter can be reset with function code 8.

The status word is FFFF (hexadecimal), if the slave is busy. Otherwise the status word is zero.

The query message contains the following information:

- Slave address
- Function code
- CRC-error check (Serial port communication only)

The response message contains the following information:

- Slave address
- Function code
- Status high
- Status low
- Event count high
- Event count low
CRC - error check (Serial port communication only)

There is no individual counter for each station connected to SYS 600, but there is only one common event counter for all the stations.

5.4.2.5 Function code 15 - Force multiple coils

This function code sets a number of commands in one Modbus message. The slave treats these commands as digital inputs and also stores them as such. This function code is equivalent to function code 5 described earlier.

The maximum number of commands that can be sent in one message is limited to 38. If the master tries to send more than 38 commands in one message, an exception response with an error code is sent to the master. The valid address range is 1…2048. Any value outside this range results in an exception response.

5.4.2.6 Function code 16 - Modify multiple registers

This function code is equivalent to function code 6 with only one difference: it handles multiple commands. On the slave's side, the commands are treated as analog input values with an object number 3000 + register address.

The maximum number of commands that can be sent in one message is 28. If more than 28 commands are sent in one message, an exception response is sent to the master. The valid address range is 1…1095. Any value outside this range results in an exception response.

5.5 Status codes

The following status codes are defined:

- 13450 SPIC_INVALID_INDEX_RANGE
  The error reply status to an invalid index value.
- 13451 SPIC_UNKNOWN_SPI_ATTRIBUTE_VALUE
  The error reply status to an invalid attribute value.
- 13455 SPIC_UNKNOWN_SPI_ATTRIBUTE
  The error reply status to an unknown attribute.
- 13459 SPIC_FCOM_COLDSTART_RECEIVED
  The status reply when the restart communication request is received. The request is function code 8 and sub-function code 1, as defined in the Modbus protocol specification.
- 13469 SPIC_ARGUMENTEXPECTED
  The error reply status to too few arguments in an attribute command.
- 13470 SPIC_TOO_MANY_ARGUMENTS
  The error reply status to too many arguments in an attribute command.
- 13484 SPIC_COMMUNICATION_WITH_CS_LOST
  The error reply status to communication with the control station being lost.
The status reply when communication to every Modbus master has been lost. Also at restart this status reply implements that a master connection is not yet received.

- **13485 SPIP_COMMUNICATION_WITH_CS_ESTABLISHED**
  The status reply when a connection to a Modbus master has been made.

- **14016 NETWUNKNOWN_DESTINATION_DEVICE**
  The status reply when MicroSCADA is trying to access an unknown device.

When used with COM 500i, only the following error codes are available:

- **13459 SPIC_FCOM_COLDSTART_RECEIVED**
  The status reply when the restart communication request is received. The request is function code 8 and sub-function code 1, as defined in the Modbus protocol specification.

- **13484 SPIP_COMMUNICATION_WITH_CS_LOST**
  The status reply when communication to every Modbus master has been lost. Also at restart this status reply implements that no master connection is yet received.

- **13485 SPIP_COMMUNICATION_WITH_CS_ESTABLISHED**
  This is the status reply when the station is running and a connection to a Modbus master has been made.

When se_applevent_used config.ini configuration is set to 4, the following status codes are also used:

- **0 STATION_OK**
  The status reply when the station is running and connected to the master.

- **13489 SPIP_STATION_STOPPED**
  The status reply when the station set to the Not In Use state.

## 5.6 Interoperability list

Functions and parameters:

- **Supported** by SYS 600 Modbus Slave
- **Not supported** by SYS 600 Modbus Slave

### 5.6.1 Network Configurations

- **Point-to-point**
- **Multipoint star**
- **Multiple point-to-point**

### 5.6.2 Physical Layer

**Electrical Interface**
5.6.3 Link Layer

Protocol Mode

- ASCII
- RTU

Message Length

255 Maximum length L (number of characters)

5.6.4 Transmission Speed (only serial)

- 300 bit/s
- 1200 bit/s
- 4800 bit/s
- 9600 bit/s
- 19200 bit/s
- 57600 bit/s

5.6.5 Transmission Settings (only serial)

Parity Check

- No parity check
- Even parity check
- Odd parity check
- Mark parity check
- Space parity check

Stop Bits

- 1 stop bit
- 2 stop bits
- 1.5 stop bits

5.6.6 Application Layer

Function Codes
## Modbus Slave Protocol

### Configuration Manual

<table>
<thead>
<tr>
<th>Command Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1&gt;</td>
<td>Read Coil Status</td>
</tr>
<tr>
<td>&lt;2&gt;</td>
<td>Read Input Status</td>
</tr>
<tr>
<td>&lt;3&gt;</td>
<td>Read Holding Register</td>
</tr>
<tr>
<td>&lt;4&gt;</td>
<td>Read Input Registers</td>
</tr>
<tr>
<td>&lt;5&gt;</td>
<td>Force Single Coil</td>
</tr>
<tr>
<td>&lt;6&gt;</td>
<td>Write Single Register</td>
</tr>
<tr>
<td>&lt;7&gt;</td>
<td>Read Exception Status</td>
</tr>
<tr>
<td>&lt;8&gt;</td>
<td>Diagnostics</td>
</tr>
<tr>
<td>&lt;9&gt;</td>
<td>Read Input Registers</td>
</tr>
<tr>
<td>&lt;10&gt;</td>
<td>Read Holding Register</td>
</tr>
<tr>
<td>&lt;11&gt;</td>
<td>Get Communication Event Counter</td>
</tr>
<tr>
<td>&lt;12&gt;</td>
<td>Get Communication Event Log</td>
</tr>
<tr>
<td>&lt;13&gt;</td>
<td>Force Single Coil</td>
</tr>
<tr>
<td>&lt;14&gt;</td>
<td>Write Single Register</td>
</tr>
<tr>
<td>&lt;15&gt;</td>
<td>Force Multiple Coils</td>
</tr>
<tr>
<td>&lt;16&gt;</td>
<td>Write Multiple Registers</td>
</tr>
<tr>
<td>&lt;17&gt;</td>
<td>Report Slave ID</td>
</tr>
<tr>
<td>&lt;18&gt;</td>
<td>Read Exception Status</td>
</tr>
<tr>
<td>&lt;19&gt;</td>
<td>Diagnostics</td>
</tr>
<tr>
<td>&lt;20&gt;</td>
<td>Read File Record</td>
</tr>
<tr>
<td>&lt;21&gt;</td>
<td>Write File Record</td>
</tr>
<tr>
<td>&lt;22&gt;</td>
<td>Mask Write Register</td>
</tr>
<tr>
<td>&lt;23&gt;</td>
<td>Read/Write Multiple Registers</td>
</tr>
<tr>
<td>&lt;24&gt;</td>
<td>Read FIFO Queue</td>
</tr>
</tbody>
</table>
## Index

### A

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address_offset</td>
<td>17, 29</td>
</tr>
<tr>
<td>Addressing schematics</td>
<td>29</td>
</tr>
<tr>
<td>Al</td>
<td>34</td>
</tr>
<tr>
<td>allowed_ip_addr_x</td>
<td>17</td>
</tr>
<tr>
<td>Analog</td>
<td></td>
</tr>
<tr>
<td>Commands</td>
<td>36, 40</td>
</tr>
<tr>
<td>Inputs</td>
<td>36, 42</td>
</tr>
<tr>
<td>Offsets</td>
<td>46</td>
</tr>
<tr>
<td>Setpoints</td>
<td>29, 40, 45</td>
</tr>
<tr>
<td>Values</td>
<td>29, 35, 38, 44</td>
</tr>
<tr>
<td>Analog Initialization</td>
<td>34</td>
</tr>
<tr>
<td>Analog Value</td>
<td>31</td>
</tr>
<tr>
<td>application_number</td>
<td>15</td>
</tr>
<tr>
<td>AV</td>
<td>31, 35, 38</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>basesystem_node_no</td>
<td>15</td>
</tr>
<tr>
<td>basesystem_station_no</td>
<td>15</td>
</tr>
<tr>
<td>Base system as Modbus master</td>
<td>25</td>
</tr>
<tr>
<td>Binary inputs</td>
<td>38</td>
</tr>
<tr>
<td>Bit number</td>
<td>37</td>
</tr>
<tr>
<td>Block number</td>
<td>29, 37</td>
</tr>
<tr>
<td>Broadcast mode</td>
<td>42–43</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>change_bit_offset</td>
<td>17</td>
</tr>
<tr>
<td>Clear counters and diagnostic register</td>
<td>47</td>
</tr>
<tr>
<td>Coil</td>
<td>30</td>
</tr>
<tr>
<td>Coil address</td>
<td>43, 46</td>
</tr>
<tr>
<td>COM 500i</td>
<td>29, 35, 50</td>
</tr>
<tr>
<td>Command procedures</td>
<td>40</td>
</tr>
<tr>
<td>Communication attributes</td>
<td>31</td>
</tr>
<tr>
<td>Communication frontend</td>
<td>28</td>
</tr>
<tr>
<td>Communication Programming Interface (CPI)</td>
<td>27</td>
</tr>
<tr>
<td>Config.ini</td>
<td>14, 44</td>
</tr>
<tr>
<td>Configuration</td>
<td>11</td>
</tr>
<tr>
<td>Configuration test</td>
<td>25</td>
</tr>
<tr>
<td>connection_watchdog</td>
<td>17</td>
</tr>
<tr>
<td>CPI application</td>
<td>27</td>
</tr>
<tr>
<td>CPI database</td>
<td>35, 41</td>
</tr>
<tr>
<td>CRC error</td>
<td>44, 47</td>
</tr>
<tr>
<td>Cross-reference information</td>
<td>29</td>
</tr>
</tbody>
</table>

### D

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_bits</td>
<td>16</td>
</tr>
<tr>
<td>Database Initialised</td>
<td>33</td>
</tr>
<tr>
<td>Data flow</td>
<td>28</td>
</tr>
<tr>
<td>DD</td>
<td>35, 39</td>
</tr>
<tr>
<td>Debug parameters</td>
<td>19</td>
</tr>
<tr>
<td>Debug window</td>
<td>20, 24</td>
</tr>
<tr>
<td>DI</td>
<td>33</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>46</td>
</tr>
<tr>
<td>Digital</td>
<td></td>
</tr>
<tr>
<td>Commands</td>
<td>36</td>
</tr>
<tr>
<td>Inputs</td>
<td>40, 42, 49</td>
</tr>
<tr>
<td>Directory</td>
<td></td>
</tr>
<tr>
<td>\progs\Modbus_Slave</td>
<td>11, 14</td>
</tr>
<tr>
<td>Discrete inputs</td>
<td>30</td>
</tr>
<tr>
<td>Double Indication</td>
<td>35–36, 39, 43</td>
</tr>
<tr>
<td>dtr_control</td>
<td>16</td>
</tr>
</tbody>
</table>

### E

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_debuts_only</td>
<td>20</td>
</tr>
<tr>
<td>Error codes</td>
<td>50</td>
</tr>
</tbody>
</table>

### F

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatal error</td>
<td>9</td>
</tr>
<tr>
<td>Fetch communication event counter</td>
<td>48</td>
</tr>
<tr>
<td>FFFF</td>
<td>48</td>
</tr>
<tr>
<td>FI</td>
<td>37, 40</td>
</tr>
<tr>
<td>flow_control</td>
<td>16</td>
</tr>
<tr>
<td>Force multiple coils</td>
<td>42, 49</td>
</tr>
<tr>
<td>Force single coil</td>
<td>42</td>
</tr>
<tr>
<td>Forcing a coil</td>
<td>45</td>
</tr>
<tr>
<td>format_analog</td>
<td>19</td>
</tr>
<tr>
<td>Free Integer</td>
<td>37</td>
</tr>
<tr>
<td>Free teXt</td>
<td>37</td>
</tr>
<tr>
<td>Function codes</td>
<td>41, 43</td>
</tr>
</tbody>
</table>

### G

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>General parameters</td>
<td>17</td>
</tr>
</tbody>
</table>

### I

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>32, 35, 38</td>
</tr>
</tbody>
</table>
# SYS 600 9.3 MicroSCADA Pro
## Modbus Slave Protocol
### Configuration Manual

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Protocol queries</td>
</tr>
<tr>
<td>17</td>
<td>Protocol addressing</td>
</tr>
<tr>
<td>41</td>
<td>Process commands</td>
</tr>
<tr>
<td>14</td>
<td>PC-NET</td>
</tr>
<tr>
<td>35</td>
<td>PC</td>
</tr>
<tr>
<td>16</td>
<td>parity</td>
</tr>
<tr>
<td>34</td>
<td>Indication Initialization</td>
</tr>
<tr>
<td>32</td>
<td>InDications</td>
</tr>
<tr>
<td>28</td>
<td>Input data</td>
</tr>
<tr>
<td>27</td>
<td>Intelligent Electronic Devices (IEDs)</td>
</tr>
<tr>
<td>50</td>
<td>Interoperability list</td>
</tr>
<tr>
<td>33</td>
<td>In Use</td>
</tr>
<tr>
<td>33</td>
<td>IU</td>
</tr>
<tr>
<td>12</td>
<td>LAN link</td>
</tr>
<tr>
<td>18</td>
<td>no_of_stns</td>
</tr>
<tr>
<td>16</td>
<td>no_of_masters</td>
</tr>
<tr>
<td>17</td>
<td>no_of_ips</td>
</tr>
<tr>
<td>15</td>
<td>ND</td>
</tr>
<tr>
<td>20</td>
<td>namedpipe_name</td>
</tr>
<tr>
<td>15</td>
<td>se_applevent_used</td>
</tr>
<tr>
<td>16</td>
<td>stop_bit</td>
</tr>
<tr>
<td>19</td>
<td>status</td>
</tr>
<tr>
<td>31</td>
<td>SPI station</td>
</tr>
<tr>
<td>37</td>
<td>Object Status</td>
</tr>
<tr>
<td>33</td>
<td>Object index</td>
</tr>
<tr>
<td>30</td>
<td>Offset parameter</td>
</tr>
<tr>
<td>33</td>
<td>OS</td>
</tr>
<tr>
<td>29</td>
<td>Output data</td>
</tr>
<tr>
<td>15</td>
<td>own_node_no</td>
</tr>
<tr>
<td>15</td>
<td>own_station_no</td>
</tr>
<tr>
<td>15</td>
<td>OWN</td>
</tr>
<tr>
<td>35</td>
<td>slave_ip_addr</td>
</tr>
<tr>
<td>36</td>
<td>SINCDAC</td>
</tr>
<tr>
<td>40</td>
<td>Single indications</td>
</tr>
<tr>
<td>43</td>
<td>SPI station</td>
</tr>
<tr>
<td>15</td>
<td>serial_port_2_in_use</td>
</tr>
<tr>
<td>15</td>
<td>serial_port_in_use</td>
</tr>
<tr>
<td>15</td>
<td>Serial port communication parameters</td>
</tr>
<tr>
<td>35</td>
<td>Signals</td>
</tr>
<tr>
<td>40</td>
<td>Status codes</td>
</tr>
<tr>
<td>36</td>
<td>Status information</td>
</tr>
<tr>
<td>44</td>
<td>stop_bit</td>
</tr>
<tr>
<td>7</td>
<td>SYS 600</td>
</tr>
<tr>
<td>31</td>
<td>SYS BASCON.COM</td>
</tr>
<tr>
<td>19</td>
<td>system self supervision</td>
</tr>
<tr>
<td>39</td>
<td>stop_bit</td>
</tr>
<tr>
<td>39</td>
<td>SYSCON 9.3</td>
</tr>
</tbody>
</table>

### R
- Read coil status .................................. 42–43
- Read holding register ............................... 42
- Read holding registers .............................. 44
- Read input registers ................................ 42, 45
- Read input status ................................... 42, 44
- Register number .................................... 29
- Remote Terminal Units (RTUs) ....................... 27
- respond also if invalid ........................... 19
- Response message .................................... 43
- Restart communication option ...................... 46
- Return bus communication error count ............ 47
- Return bus message count ........................... 47
- Return query data .................................... 46
- Return slave message count ......................... 48
- Return slave no response count .................... 48
- RP 570 station ....................................... 31
- RTU station ........................................... 12

### S
- SA ...................................................... 15, 25
- se_applevent_used .................................. 19
- serial_port_2_in_use ................................ 15
- serial_port_in_use .................................... 15
- Serial port communication parameters .......... 15
- Signals ............................................... 35
- SINCDAC .............................................. 40
- Single indications ................................... 36, 43
- slave_ip_addr ........................................ 16
- SPI station ........................................... 31, 37
- Status codes ......................................... 10, 49
- Status information ................................... 39
- stn_no ................................................ 19
- SYNC .................................................. 19
- Time stamp ......................................... 31

### T
- tcp_ip_port ......................................... 16
- tcp_ipadd ............................................ 15
- tcpip_1_in_use ...................................... 15
- tcpip_2_in_use ...................................... 15
- Time stamp ......................................... 31, 39

---

**Note:** The table above is a representation of the content from the given page of the document. The page numbers are for reference only and may not correspond to the actual page layout in the document.
<table>
<thead>
<tr>
<th>V</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Write multiple registers</td>
</tr>
<tr>
<td>................................</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write single register</td>
</tr>
<tr>
<td>................................</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>