SOFTWARE MANUAL

PCO Library
# Table of contents

1  AC500 Process Control Object (PCO) Library............................................................................................................. 3
   1.1  PCO Library - System Technology.................................................................................................................. 3
       1.1.1  Introduction................................................................................................................................ 3
       1.1.2  Installation.................................................................................................................................. 4
       1.1.3  Prerequisites................................................................................................................................. 7
       1.1.4  Configuration................................................................................................................................. 8
       1.1.5  Engineering Workflow.............................................................................................................. 20
       1.1.6  Capacity and Performance....................................................................................................... 22
   1.2  PCO Library - Function Block Description (V2)............................................................................................ 25
       1.2.1  Scope and Structure of this Document.................................................................................... 25
       1.2.2  Process Control Object (PCO) Library..................................................................................... 26
2  Index.................................................................................................................................................................... 121
1 AC500 Process Control Object (PCO) Library

1.1 PCO Library - System Technology

1.1.1 Introduction

800xA Connect in combination with AC500 Process Control Object (PCO) library allows to easily integrate AC500 into 800xA for process control:

The AC500 PCO Library contains function blocks for Process Control Objects like motors, valves, PID controllers, alarms, etc. Those function blocks are integrated into 800xA via OPC.

A corresponding 800xA aspect object allows control and monitoring of the Process Control Objects with 800xA User Interface like faceplates and graphic elements:
1.1.2 Installation

For integration of AC500 into 800xA it is recommended to have at least two nodes:

- AC500 engineering node with Automation Builder and PCO library
- 800xA engineering node for engineering and operation of 800xA. The OPC server for AC500 ("Codesys OPC Server V3.5") must be installed on this node for connectivity with AC500.

It is also possible to install the AC500 Automation Builder on the 800xA engineering node,

but this document describes the general case with two nodes.

1.1.2.1 Install Automation Builder on AC500 Engineering Node

Latest Automation Builder installation files and instructions can be found at https://new.abb.com/plc/automationbuilder.

During installation the PCO library can be chosen as an option.

1.1.2.2 Install AC500 PCO Library on AC500 Engineering Node

If the PCO library is not yet installed from the beginning it can be installed using the Automation Builder Installation Manager:

1.1.2.3 Install AC500 Connect on 800xA node

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.
1.1.2.4 Install AC500 OPC Server on 800xA Node

The AC500 OPC Server ("Codesys OPC Server 3.5") must be installed on the 800xA engineering node. The AC500 OPC server is part of the Automation Builder installation, but it is not required to install the full Automation Builder on the 800xA engineering node.

Latest Automation Builder installation files can be found in http://new.abb.com/plc/automationbuilder.

Checkpoint 1

1. Before starting the installation ensure that all OPC clients from previous versions are closed:
   - "ABB OPC Tunnel"
   - Gateway (CODESYS gateway server)

   This can be checked in the Windows Task Manager.

   The following processes must have disappeared:
   - "Gateway.exe",
   - "CoDeSysOPC.exe",
   - "WinCoDeSysOPC.exe" and
   - "OCTsvc.exe"

   If not:
   - End the processes with the Windows Task Manager.
   - Stop "ABB OPC Tunnel Windows Component Service" in Services (local).

2. Start the Automation Builder installation.
3. Select [Install Additional Tools] and install the “CODESYS OPC Server 3.5” which includes the “AC500 Gateway Drivers”: 

4. Alternatively, you can select [Prepare Offline Installation Additional Tools] to create an USB-stick containing the “CODESYS OPC Server 3.5” and use this for installation on the 800xA engineering node.

5. During installation on the 800xA engineering node select all features.
1.1.3 Prerequisites

1.1.3.1 800xA License

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.1.3.2 AC500 License

A separate license is not required for the PCO library.
1.4 Configuration

This chapter describes the configuration of AC500 engineering node and 800xA engineering node.

1.4.1 Configure AC500 OPC Server

Communication of AC500 to 800xA is done through OPC. Therefore it is required to install the AC500 OPC server (see Chapter 1.2.4 “Install AC500 OPC Server on 800xA Node” on page 5) and to configure it accordingly.

Basic configuration workflow for AC500 OPC Server: The definition of the items (symbols) to be exchanged over OPC DA are stored in the symbol file “*.sdb”. The symbol file is generated by the AC500 engineering tool Automation Builder which is running on the AC500 engineering node. When downloading the application to the AC500 PLC the “*.sdb” file is stored on the user disk of the PLC. Finally, the OPC Server running on the 800xA engineering node uploads the “*.sdb” file in order to configure the tags accordingly.

This chapter describes how to setup the OPC connection on the AC500 engineering node as well as on the 800xA engineering node.
1.1.4.1.1 AC500 Engineering Node

It is assumed that a simple AC500 project was configured with an AC500 engineering node running the Automation Builder. For more information please check chapter 1.1.5.1 “Create Function Blocks in AC500 V2” on page 20.

As a quick start the simple motor example “PCO_Motor_Demo_AB223.project” can be used which comes with the PCO library package, see folder C:\Users\Public\Documents\AutomationBuilder\Examples\PS573-PCO.

The following chapter describes the configuration steps to prepare the AC500 for OPC communication to the 800xA engineering node.

Configure Symbol File in Automation Builder

1. Open Automation Builder and start the application of the project which will open CODESYS.
2. Select “Options” in menu “Project”.
3. Configure symbol file according to the instructions in the Automation Builder help: see

For the example “PCO_Motor_Demo_AB223.project” it should look like this:
Create and Download Symbol File

1. Follow the instruction in the Automation Builder help: see

Checkpoint 2

2. Verify that the symbol file is downloaded to the PLC by opening the “PLC-Browser” in “CODESYS” (online mode) and entering “fdir userdisk”:

![SCADA Browser Screenshot](image-url)
1.1.4.1.2 800xA Engineering Node

Configure OPC Server

1. Start “OPC Configurator”.

2. Default values of the server need not to be changed.

   ```markdown
   “Update Rate” may should not be “0 ms”! The default value of “200 ms” is suitable value of many applications. The adjustment for the update rate depends on the number of symbols (variables). For a big number of symbols it would be better to increase the update rate.
   ```

3. Configure the PLC1.

4. Choose “Interface” “GATEWAY”.
   This is the V2 Gateway which communicates with the AC500 V2.
   - “Project name” can be empty.
   - Increase “Buffer Size [Byte]” to “4800”
   - The checkboxes “Active”, “Motorola Byteorder” must be checked.
   - The checkbox “Enable logging [Defaultevents]” allows a later diagnosis.

[Communication Parameters]

**Address** must be the IP address of the connected Ethernet port of AC500.

6. Click [Gateway] and configure "Tcp/Ip".

[Communication Parameters: Gateway]

7. Final configuration looks like this:

[OPCConfig]
8. Save the configuration.

Read OPC Data with Matrikon Test Client (optional)

This chapter describes how to install and configure the Matrikon OPC test client. This is an optional step but it is recommended to test the basic OPC communication before doing any 800xA specific configuration.

The AC500 must be connected to the 800xA engineering node and its IP address must be in the same LAN.

Checkpoint 3

1. Test if you can ping the PLC from 800xA engineering node:

```
C:\Windows\System32\cmd.exe

C:\Windows [Version 6.3.9600] (c) 2013 Microsoft Corporation. All rights reserved.
C:\Windows\System32>ping 192.168.20.10
Pinging 192.168.20.10 with 32 bytes of data:
Reply from 192.168.20.10: bytes=32 time=1ms TTL=255
Reply from 192.168.20.10: bytes=32 time<1ms TTL=255
Reply from 192.168.20.10: bytes=32 time<1ms TTL=255
Reply from 192.168.20.10: bytes=32 time<1ms TTL=255
Ping statistics for 192.168.20.10:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
  Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
C:\Windows\System32>
```

2. Install an OPC test client, for example from Matrikon.

More information on test clients can be found in the Automation Builder help: see
3. Start the OPC test client (here: “Matrikon OPC Explorer”) and connect the “CoDeSys.OPC.DA”:

4. After clicking [eConnect] the “Tray Icon” of the CODESYS gateway turns from idle to active.

Furthermore, the sdb file is uploaded from the PLC to the following Gateway folder: C:\ProgramData\Gateway Files\Upload
5. Add tags.

6. Watch the values of the added tags.

Checkpoint 5

7. Quality of the tag must be “Good”
Register OPC Server as System Service

For use with 800xA it is important that the OPC server runs with Session ID 0 (like all other 800xA services).

Therefore OPC server must be registered as service.

1. Start a command prompt as administrator.

2. Go to the CoDeSysOPC V3 installation folder.
3. Unregister the OPC server with "WinCoDeSysOPC/UnRegServer".
4. Register the OPC server as system service with "WinCoDeSysOPC/Service"

Checkpoint 6

5. During this procedure there should be no errors, terminal should look like this:

---

Same information in Automation Builder online help: see
6. Open Task Manager and select “CoDeSysOPCDAService” from the “Service” menu. Right-click the “CoDeSysOPCDAService” and select “Open Services”:

7. Double-click “CoDeSysOPCDAService” and configure properties, logon with 800xAService account.

   “800xAService” in this example, maybe different!

8. Restart the service (right-click).

**Checkpoint 7**

9. Now the process runs with *User name* “800xAService” and *Session ID* “0”.
1.1.4.1.3 Troubleshooting

In case of any problems please check the potential issues in the Automation Builder help: Potential Issues.
1.1.4.2 Create PLC Generic Control Network Object
This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.1.4.3 Configure PLC Connect Services
This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.1.4.4 Create AC500 Controller and GCN configuration
This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.
1.1.5 Engineering Workflow

Engineering is done in 3 steps:

1. Create application in AC500 using the AC500 PCO library.
2. Create equivalent structure in “Bulk Data Manager”.
3. Populate PLC objects to control structure using the 800xA PLC object library.

1.1.5.1 Create Function Blocks in AC500 V2

1. For general introduction to AC500 configuration please refer to the chapter Getting Started.
2. Create a new project: Project Handling
3. Transfer data to CODESYS: Data Transfer
4. Program your application in Codesys V2

1.1.5.2 Function Blocks of PCO Library

The PCO library contains the AC500 function blocks (motors, valves, …) which can be integrated into 800xA.

The function blocks can be controlled and monitored by 800xA during operation.

Installation of the PCO library is described in % Chapter 1.1.2.2 “Install AC500 PCO Library on AC500 Engineering Node” on page 4.

All function blocks are described in % Chapter 1.2 “PCO Library - Function Block Description (V2)” on page 25.
1.1.5.3 Create Instances using Bulk Data Manager

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.
1.1.6 Capacity and Performance

1.1.6.1 AC500 Function Block Performance

The PCO (Process Control Object) Library is usable on the whole range of AC500 platform including AC500-eCo PLC’s.

The maximum number of function blocks per CPU is limited:

- By the available program memory for each CPU type.
- By the available speed for each CPU type.
- By the PLC cycle time which needs to be set to achieve desired CPU load.

The table below portrays the performance overview of various CPU’s. It shows two typical configurations as examples.

These examples can be used by user to understand the performance of PLC’s and choose the PLC based on application.

1.1.6.1.1 Small Configuration

It uses 39 instances of function blocks. The various blocks used are:

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Number of Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCO_BINSET</td>
<td>1</td>
</tr>
<tr>
<td>PCO_BIN</td>
<td>12</td>
</tr>
<tr>
<td>PCO_ALARM</td>
<td>6</td>
</tr>
<tr>
<td>PCO_ANASET</td>
<td>1</td>
</tr>
<tr>
<td>PCO_ANA</td>
<td>1</td>
</tr>
<tr>
<td>PCO_ANAALM</td>
<td>9</td>
</tr>
<tr>
<td>PCO_ANALIM</td>
<td>1</td>
</tr>
<tr>
<td>PCO_MOT</td>
<td>3</td>
</tr>
<tr>
<td>PCO_VALV</td>
<td>2</td>
</tr>
<tr>
<td>PCO_MOTCON</td>
<td>1</td>
</tr>
<tr>
<td>PCO_PIDCON</td>
<td>1</td>
</tr>
<tr>
<td>PCO_VALVCON</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>39</strong></td>
</tr>
</tbody>
</table>

When these instances of function blocks were downloaded the memory coverage in various PLC’s is shown in the following table:

<table>
<thead>
<tr>
<th>PLC</th>
<th>Required User Data Memory (% used from total available)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM564-ETH</td>
<td>7730 → 75 %</td>
</tr>
<tr>
<td>PM573-ETH</td>
<td>7730 → 3 %</td>
</tr>
<tr>
<td>PM591-ETH</td>
<td>7730 → 0.18 %</td>
</tr>
<tr>
<td>PM595-4ETH</td>
<td>7730 → 0.05 %</td>
</tr>
</tbody>
</table>
The cycle time versus the CPU load for various PLC’s are shown in table below. The user needs to select the cycle time based on desired CPU load. Cycle time and CPU load are inversely proportional and are depicted in the following figure. It is not desirable for the CPU load to be 100 % and hence the cycle time needs to be chosen likewise.

![Performance graphs](image)

1.1.6.1.2 Medium Configuration

It uses 49 instances of function blocks. The various blocks used are:

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Number of Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCO_BINTSET</td>
<td>3</td>
</tr>
<tr>
<td>PCO_BIN</td>
<td>6</td>
</tr>
<tr>
<td>PCO_ALARM</td>
<td>2</td>
</tr>
<tr>
<td>PCO_ANASET</td>
<td>2</td>
</tr>
<tr>
<td>PCO_ANA</td>
<td>1</td>
</tr>
<tr>
<td>PCO_ANAALM</td>
<td>9</td>
</tr>
<tr>
<td>PCO_ANALIM</td>
<td>1</td>
</tr>
<tr>
<td>PCO_MOT</td>
<td>2</td>
</tr>
<tr>
<td>PCO_VALV</td>
<td>1</td>
</tr>
<tr>
<td>PCO_MOTCON</td>
<td>7</td>
</tr>
<tr>
<td>PCO_PIDCON</td>
<td>1</td>
</tr>
<tr>
<td>PCO_VALVCON</td>
<td>14</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>49</strong></td>
</tr>
</tbody>
</table>

AC500 Process Control Object (PCO) Library
PCO Library - System Technology > Capacity and Performance
When these instances of function blocks were downloaded the memory coverage in various PLC’s is shown in the following table:

<table>
<thead>
<tr>
<th>PLC</th>
<th>Required User Data Memory (%) used from total available</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM573-ETH</td>
<td>18258 → 7 %</td>
</tr>
<tr>
<td>PM591-ETH</td>
<td>18258 → 0.44 %</td>
</tr>
<tr>
<td>PM595-4ETH</td>
<td>18258 → 0.11 %</td>
</tr>
</tbody>
</table>

Here the program could not be downloaded to PM564-ETH because of the less memory available (Program Code Memory Max - 131072 Bytes) and here it is 18258 Bytes and hence PM573-ETH, PM591-ETH and PM595-4ETH were used for testing.

The cycle time versus the CPU load for various PLC’s are shown in table below. The user needs to select the cycle time based on desired CPU load. Cycle time and CPU load are inversely proportional and are depicted in the following figure. It is not desirable for the CPU load to be 100 % and hence the cycle time needs to be chosen likewise.

![Performance graphs](chart.png)
1.1.6.1.3 Memory Usage per Function Block

The table below shows the memory size of each function block of the PCO library.
This information can be used by the user to select the CPU type based on its application.

<table>
<thead>
<tr>
<th>Component</th>
<th>Required User Program Memory (Bytes) (Max - 524288)</th>
<th>Required User Data Memory (Bytes) (Max - 65536)</th>
<th>Size of individual FB (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of 1 Function Block Instance</td>
<td>Number of 5 Function Block Instances</td>
<td></td>
</tr>
<tr>
<td>Project no program</td>
<td>4338 (0 %)</td>
<td>1958 (2 %)</td>
<td></td>
</tr>
<tr>
<td>Project with Library</td>
<td>4354 (0 %)</td>
<td>1962 (2 %)</td>
<td></td>
</tr>
<tr>
<td>PCO_BINSET</td>
<td>5070 (0 %)</td>
<td>1976 (3 %)</td>
<td>14</td>
</tr>
<tr>
<td>PCO_BIN</td>
<td>5174 (0 %)</td>
<td>1991 (3 %)</td>
<td>29</td>
</tr>
<tr>
<td>PCO_ALARM</td>
<td>5206 (0 %)</td>
<td>1991 (3 %)</td>
<td>29</td>
</tr>
<tr>
<td>PCO_ANASET</td>
<td>5018 (0 %)</td>
<td>1978 (3 %)</td>
<td>16</td>
</tr>
<tr>
<td>PCO_ANA</td>
<td>5174 (0 %)</td>
<td>1974 (3 %)</td>
<td>12</td>
</tr>
<tr>
<td>PCO_ANAALRM</td>
<td>19522 (3 %)</td>
<td>2166 (3 %)</td>
<td>204</td>
</tr>
<tr>
<td>PCO_ANALIM</td>
<td>12498 (2 %)</td>
<td>2078 (3 %)</td>
<td>116</td>
</tr>
<tr>
<td>PCO_MOT</td>
<td>10922 (2 %)</td>
<td>2288 (3 %)</td>
<td>326</td>
</tr>
<tr>
<td>PCO_VALV</td>
<td>11402 (2 %)</td>
<td>2326 (3 %)</td>
<td>364</td>
</tr>
<tr>
<td>PCO_MOTCON</td>
<td>30814 (5 %)</td>
<td>2772 (4 %)</td>
<td>810</td>
</tr>
<tr>
<td>PCO_PIDCON</td>
<td>25310 (4 %)</td>
<td>2468 (3 %)</td>
<td>506</td>
</tr>
<tr>
<td>PCO_VALVCON</td>
<td>26686 (5 %)</td>
<td>2550 (3 %)</td>
<td>588</td>
</tr>
</tbody>
</table>

The percentages in the bracket denote the memory consumed out of the total available. Tested using PM566-ETH.

1.1.6.2 800xA Performance

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.2 PCO Library - Function Block Description (V2)

1.2.1 Scope and Structure of this Document

The purpose of this library description is to explain different components of the Process Control Object (PCO) library.
1.2.2 Process Control Object (PCO) Library

Process Control Object (PCO) Library is developed for use in any process application. It includes function blocks for controlling motors, valves and measurements. There are no prerequisites.

Components of the PCO Library

The PCO library contains the following function blocks and structures:
1.2.2.1 Function Blocks
1.2.2.1.1 Controllers

PCO_MOTCON

This function block is designed for controlling a variable speed drive motor. The function block is similar to PCO_MOT (Fixed speed motor controller), but in addition to PCO_MOT the function block has a built in PID controller used for speed control of the motor.

The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.

- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.

- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.
To make the controller follows various setpoints, the following variables need to be set:

The values set below are default values, the user can change based on the application.

### Settings in SCADA side

<table>
<thead>
<tr>
<th>Data type</th>
<th>When the controller must follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Setpoint</td>
<td>External Setpoint</td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Settings in FB side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Controller Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Setpoint</td>
<td>External Setpoint</td>
</tr>
<tr>
<td>EN.AUTO</td>
<td>BOOL</td>
</tr>
<tr>
<td>SP_EXT</td>
<td>INT</td>
</tr>
<tr>
<td>TRACK</td>
<td>BOOL</td>
</tr>
<tr>
<td>TRACK.SP</td>
<td>INT</td>
</tr>
<tr>
<td>EN_ACC</td>
<td>BOOL</td>
</tr>
<tr>
<td>EN_DEC</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.

The following variable needs to be configured to set the controller to manual mode.

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Controller Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>No settings needs to be done</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The manual positioning can be done in two ways, both controlled from the SCADA faceplate.

1. Stepwise 1% up / down or 5% up / down.
   
   This is manually set by the following variables on the SCADA faceplate:
   
   ● SCADA.CMDUP1
   ● SCADA.CMDUP5
   ● SCADA.CMDDW1
   ● SCADA.CMDDW5
   
   The controller output varies based on the command given.

2. Download a new position. Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.

If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

> Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_START and CMD_STOP can be configured. E.g. if CMD_START and CMD_STOP should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.)

> The motor can be switched to automatic mode or manual mode, if EN_AUTO input is TRUE in the function block.
## Start / Stop Motor in Auto Mode

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Set Motor to Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Rising edge needs to be given.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Start / Stop Motor in Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
<td>The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.</td>
</tr>
<tr>
<td>EN_START</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>EN_STOP</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>AUTO_CMD_START</td>
<td>BOOL</td>
<td>Rising edge needs to be given to AUTO_CMD_START input of FB to start the motor.</td>
</tr>
<tr>
<td>AUTO_CMD_STOP</td>
<td>BOOL</td>
<td>Rising edge needs to be given to AUTO_CMD_STOP input of FB to stop the motor.</td>
</tr>
</tbody>
</table>

## Set Motor to Manual Mode

The motor is forced to manual mode. `EN_AUTO = FALSE`.

The motor can be operated in manual mode if `EN_AUTO` is either `TRUE` or `FALSE`.

The difference is that if `EN_AUTO` is `TRUE`, the switching between Auto and Manual is possible, else the motor and controller is both to manual.

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Set Motor to Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Set Motor to Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
<td>Rising edge of SCADA.CMDMAN input to reset auto mode if it was set before</td>
</tr>
</tbody>
</table>
### Settings in FB side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Start / Stop Motor in Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>EN_START</td>
<td>BOOL</td>
</tr>
<tr>
<td>EN_STOP</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Settings in SCADA side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Start / Stop Motor in Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMDSTR</td>
<td>BOOL</td>
</tr>
<tr>
<td>CMDSTP</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

In automatic mode the motor can be started/stopped using the function block inputs (AUTO_CMD_START/AUTO_CMD_STOP).

In manual mode the motor can be started and stopped from the SCADA faceplate.

The function block includes supervision of the feedback signals RUN and STOP from the motor. If the motor fails to run or stop within the supervision time INHIB_FCT_ERR_TIME, it will generate a functional error.

If the function block input RDY is FALSE, then an external error is generated. This function block can be used in combination with the object type MOTCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

### Examples

The function block is used for controlling a variable speed drive motor, which in turn can be used for controlling the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Pump
- Main power supply, circuit breaker (or similar)

Manual mode:

- Pump controlled by the operator.

Automatic mode:

- Pump controlled by the AC500 controller.
PCO MOTCON
Manual Mode
Example

Tank
Level measurement
Drain

AC500 Program
- SP_EXT
- PV
- TRACK
- TRACK_SP
- BAL
- BAL_SP
- REVACT
- EN_AUTO
- AUTO_CMD_START
- AUTO_CMD_STOP
- EN_START
- EN_STOP
- EN_ACC
- EN_DEC
- EMEY_START
- EMEY_STOP
- ACT_POS
- RUN
- STOP
- RDY
- REMOTE
- INHIB_ERR
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- BEHAVE_ON_ERROR
- OUTPUT_HANDLING
- INHIB_FCT_ERR_TIME
- RAMP_TIME
- RAMP_ACTIVE
- SCADA

Workstation

Main power supply of the pump OK

Termister fault from the pump motor

Speed controlled pump
- Speed Setpoint
- Feedback Stopped
- Feedback Running
- Start Command
- Stop Command

SCADA Interface
Start/Stop and SP
Scaling of the SCADA Parameters

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to analog output of the AC500 I/Os.

If the setpoint and the process value do not have equal scaling, there is a need for scaling the SCADA.KP_PAR parameter to achieve the right proportional gain.
### Scaling of the Parameter SCADA.KP_PAR

The scaling of the proportional coefficient (SCADA.KP_PAR) is dependant on the scaling of the process value (PV).

The PV typically comes from the function block PCO_ANAALM.

The process value, e.g. 0 bar ... 16 bar is rescaled to the value 0 ... 16000 (SCALE_MIN - SCALE_MAX).

The scaling of the KP [%] (RAWMAX in SCADA) is then calculated by the formula:

\[
\text{KP [%]} = \frac{\text{PI out max} - \text{PI out min}}{\text{PV max} - \text{PV min}} \times 100 \%
\]

If the wanted range of SCADA.KP_PAR in SCADA (RNGMAX) is chosen to be 100 %, the RAWMAX in SCADA is calculated to be:

RAWMAX in SCADA = \[\frac{27648 - 0}{16000 - 0}\] x 100 % = 345

RNGMAX = SCADA.Signal Configuration.Range.High limit

RAWMAX = SCADA.Signal Configuration.Range.High limit in PLC

### Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.
Input Description

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP_EXT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

External setpoint.
Calculated setpoint or output from another PID controller (e.g. PCO_PIDCON) in a cascade control system.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>PV</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

Process value (PV) to be controlled.
**TRACK**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking mode.
PCO_MOTCON can be set in Tracking mode if the value of TRACK is TRUE.
Output of the controller will be set to TRACK_VALUE.
Tracking mode has higher priority compared to balancing mode.
TRACK is independent of operation mode of PCO_MOTCON.

**TRACK_SP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking setpoint.
Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.

**BAL**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing mode.
PCO_MOTCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.
Output of the controller will be set to BAL_SP.
BAL is independent of operation mode of PCO_MOTCON.

**BAL_SP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing setpoint.
Output of the controller will be set to BAL_SP if value of BAL is TRUE.

**REVACT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Reverse Action
Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.
Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

**EN_AUTO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable automatic mode of the PCO_MOTCON.
PCO_MOTCON can be set in automatic mode if value of EN_AUTO is TRUE.
PCO_MOTCON is forced in manual mode if value of EN_AUTO is FALSE.
### AUTO_CMD_START

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force the motor to start if it is in automatic mode.

E.g. AUTO_CMD_START could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

### AUTO_CMD_STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force the motor to stop if the actuator is in automatic.

AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

### EN_START

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable start of the motor.

Motor can be started if value of EN_START is TRUE.

Motor cannot be started if value of EN_START is FALSE.

EN_START is independent of operation mode of PCO_MOTCON.

EN_START has no effect in case of emergency start of motor EMCY_START.

EN_START has no effect on a motor already running.

### EN_STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable stop of the motor.

Motor can be stopped if value of EN_STOP is TRUE.

Motor cannot be stopped if value of EN_STOP is FALSE.

EN_STOP is independent of operation mode of PCO_MOTCON.

EN_STOP has no effect in case of emergency stop of motor EMCY_STOP.

EN_STOP has no effect on an already stopped motor.

### EN_ACC

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable increase of controller output.

Controller can increase output if value of EN_ACC is TRUE.

Independent of operation mode of PCO_MOTCON.
### EN_DEC

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable decrease of controller output.
Controller can decrease output if value of EN_DEC is TRUE.
Independent of operation mode of PCO_MOTCON.

### EMCY_START

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Emergency start of motor.
Motor will start if value of EMCY_START is TRUE and value of EMCY_STOP is FALSE.
If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE the motor will stop.
Independent of operation mode of PCO_MOTCON.

### EMCY_STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Emergency stop of motor.
Motor will stop if value of EMCY_STOP is TRUE.
If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE the motor will stop.
Independent of operation mode of PCO_MOTCON.

### ACT_POS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback speed of motor.

### RUN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback signal from the motor.
Motor is running if the value of RUN is TRUE.
Independent of operation mode of PCO_MOTCON.

### STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback signal from the motor.
Motor is stopped if the value of STOP is TRUE.
Independent of operation mode of PCO_MOTCON.
**RDY**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Motor is ready for operation.
Motor is ready for operation if value of RDY is TRUE.
Value of RDY is FALSE results in an external error.
Independent of operation mode of PCO_MOTCON.

**REMOTE**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Allow motor to be controlled from SCADA.
Value of REMOTE is TRUE, control from SCADA and function block is enabled.
Value of REMOTE is FALSE, motor is controlled only from function block.
PCO_MOTCON will align the block according to feedback signals.
Independent of operation mode of PCO_MOTCON.

**INHIB_ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of alarms.
If value of INHIB_ERR is TRUE all alarms from PCO_MOTCON are suppressed.
Independent of operation mode of PCO_MOTCON.

**EXT_ACK**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

External acknowledge of alarms.
If value of EXT_ACK is TRUE all alarms from PCO_MOTCON are acknowledged.
Independent of operation mode of PCO_MOTCON.

**Alarm_1**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 1.
If value of ALARM_1 is TRUE Alarm_1 is active.
Independent of operation mode of PCO_MOTCON.

**Alarm_2**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 2.
If value of ALARM_2 is TRUE Alarm_2 is active.
Independent of operation mode of PCO_MOTCON.
### Alarm_3

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 3.
If value of ALARM_3 is TRUE Alarm_3 is active.
Independent of operation mode of PCO_MOTCON.

### Alarm_4

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 4.
If value of ALARM_4 is TRUE Alarm_4 is active.
Independent of operation mode of PCO_MOTCON.

### Alarm_5

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 5.
If value of ALARM_5 is TRUE Alarm_5 is active.
Independent of operation mode of PCO_MOTCON.

### Alarm_6

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 6.
If value of ALARM_6 is TRUE Alarm_6 is active.
Independent of operation mode of PCO_MOTCON.

### BEHAVE_ON_ERROR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>0</td>
<td>0 ... 3</td>
<td>-</td>
</tr>
</tbody>
</table>

Actions that need to be executed by the FB at the event of error as set by the user.
The user can set a range of values from 0 ... 3.

BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).
BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.
BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.
BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto/Manual) of PCO_MOTCON.
### OUTPUT_HANDLING

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>0</td>
<td>0 ... 2</td>
<td>-</td>
</tr>
</tbody>
</table>

Behavior of command outputs, CMD_START / CMD_STOP.
The user can set a range of values from 0 ... 2
- OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.
- OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP)) to remain active at RUN or STOP feedback.
- With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_MOTCON.

### INHIB_FCT_ERR_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#5s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Maximum delay time from command to response from process.
If response is not received within this time limit, a function error will be generated.

### RAMP_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#30s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Actuator ramp time from 0 % ... 100 %

### RAMP_ACTIVE

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.
The value RAMP_ACTIVE is FALSE activates no ramp.
Output Description

| SP_EXT | CMD_START |
| PV | CMD_STOP |
| TRACK | OUT |
| TRACK_SP | OUT_MAX |
| BAL | OUT_MIN |
| BAL_SP | ERR |
| REVACT | ERNO |

**CMD_START**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output start, to be connected to hardware output.

**CMD_STOP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output stop, to be connected to hardware output.

**OUT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

Controller output to the motor to attain the desired speed of the motor.
<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Output max limit reached, controller at max limit.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Output min limit reached, controller at min limit.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, including:
- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_START
- EMCY_STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number
Output provides an error identifier if an invalid value was applied to an input.
ERNO always must be considered together with the output ERR.
The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”.
### Input/Output Description

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDSTR</td>
<td>BOOL</td>
<td>Start command in Manual mode</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDSTP</td>
<td>BOOL</td>
<td>Stop command in Manual mode</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Auto command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
<td>Manual command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDRES</td>
<td>BOOL</td>
<td>Acknowledge active alarms</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.MAUTO</td>
<td>BOOL</td>
<td>Motor in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.IN</td>
<td>BOOL</td>
<td>Motor is running</td>
<td>Output</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.OUT</td>
<td>BOOL</td>
<td>Motor is stopped</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.REMOTE</td>
<td>BOOL</td>
<td>Motor can be controlled from OS</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.READY</td>
<td>BOOL</td>
<td>Motor is ready for operation</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.FNCERR</td>
<td>BOOL</td>
<td>Functional error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.EXTERR</td>
<td>BOOL</td>
<td>External error (is generated when the motor is not ready)</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL1</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 1</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL2</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 2</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL3</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 3</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL4</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 4</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL5</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 5</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL6</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 6</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg1 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_MOTCON</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg2 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_MOTCON</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CM_CMDON</td>
<td>BOOL</td>
<td>Manual command for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
<td>Auto command to set the internal setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
<td>Auto command to set the external setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDUP5</td>
<td>BOOL</td>
<td>Manual command to increase output by 5 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDUP1</td>
<td>BOOL</td>
<td>Manual command to increase output by 1 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW1</td>
<td>BOOL</td>
<td>Manual command to decrease output by 1 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW5</td>
<td>BOOL</td>
<td>Manual command to decrease output by 5 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CAUTO</td>
<td>BOOL</td>
<td>Controller is in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPIACT</td>
<td>BOOL</td>
<td>Internal setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPXACT</td>
<td>BOOL</td>
<td>External setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CTACT</td>
<td>BOOL</td>
<td>Tracking setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_H</td>
<td>BOOL</td>
<td>Process value at high limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_L</td>
<td>BOOL</td>
<td>Process value at low limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_ERR</td>
<td>BOOL</td>
<td>Process value error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.ERROR</td>
<td>BOOL</td>
<td>For future use</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OUT_PAR</td>
<td>INT</td>
<td>Manual position setpoint</td>
<td>Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scale 0 ... 10000</td>
<td></td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
<td>Internal setpoint value</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
<td>Proportional coefficient (gain) % of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
<td>Time constant for D-part of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
<td>Time constant for integration of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
<td>Output high limit (0 ... 10000) of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
<td>Output low limit (0 ... 10000) of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OUT_VALUE</td>
<td>INT</td>
<td>Output from the controller</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_VALUE</td>
<td>INT</td>
<td>Process value to be controlled</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPX_VALUE</td>
<td>INT</td>
<td>External setpoint value</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.ACT_VALUE</td>
<td>INT</td>
<td>Actual speed of the motor, scaling as input ACT_POS (0 ... 10000)</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.LOAD_VALUE</td>
<td>INT</td>
<td>Load value of the motor, additional value to be shown in SCADA</td>
<td>Output</td>
</tr>
</tbody>
</table>

*) structure described separately

### SCADA.OSMsg1

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_MOTCON.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Motor is running</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Motor is stopped</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Motor is starting</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Motor is stopping</td>
</tr>
<tr>
<td>Bit 4</td>
<td>External error (Not ready)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Functional error</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Motor released for start</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Motor released for stop</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Local operation (Not remote)</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Common alarm (External alarm + Functional alarm + EMCY_START/EMCY_STOP + ALARM_1 ... ALARM_6)</td>
</tr>
<tr>
<td>Bit 10</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 11</td>
<td>Motor in automatic mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Motor not released for automatic mode</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Motor released for increase speed</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Motor released for decrease speed</td>
</tr>
<tr>
<td>Bit</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Emergency In</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Emergency Out</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Internal setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 3</td>
<td>External setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Track setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Auxiliary alarm 1</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Auxiliary alarm 2</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Auxiliary alarm 3</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Auxiliary alarm 4</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Auxiliary alarm 5</td>
</tr>
<tr>
<td>Bit 10</td>
<td>PID controller in automatic mode</td>
</tr>
<tr>
<td>Bit 11</td>
<td>PID controller in manual mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Auxiliary alarm 6</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 15</td>
<td>PID Controller</td>
</tr>
</tbody>
</table>

**PCO PIDCON**

This is a standard PID controller that can be used to control any component that has an analog positioning option.
The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.

- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.

- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.

To make the controller follows various setpoints, the following variables need to be set:

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>When the controller must follow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Internal Setpoint</td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
<td>10000</td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
<td>Needs to be set</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>When the controller must follow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Internal Setpoint</td>
</tr>
<tr>
<td>EN_AUTO</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>SP_EXT</td>
<td>INT</td>
<td>-</td>
</tr>
<tr>
<td>TRACK</td>
<td>BOOL</td>
<td>-</td>
</tr>
<tr>
<td>TRACK_SP</td>
<td>INT</td>
<td>-</td>
</tr>
</tbody>
</table>

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.
The manual positioning can be done in two ways, both controlled from the SCADA faceplate.

1. Stepwise 1 % up / down or 5 % up / down.
   This is manually set by the following variables on the SCADA faceplate:
   - SCADA.CMDUP1
   - SCADA.CMDUP5
   - SCADA.CMDDW1
   - SCADA.CMDDW5
   The controller output varies based on the command given.

2. Download a new position. Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.

If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

6 different alarms can be connected to the function block (ALARM_1 → ALARM_6, Torque switch etc.).

The valve can be switched to automatic or manual mode from the SCADA faceplate, if the input EN_AUTO = TRUE.

This function block can be used in combination with the object type PIDCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Examples

The function block is used for controlling a PID controller which in turn controls the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Pump

Manual mode:
- Pump controlled by the operator.

Automatic mode:
- Pump controlled by the AC500 controller.
PCO_PIDCON
Manual Mode
Example

Speed controlled pump

Feedback Stopped
Feedback Running

Start and stop of the pump is handled by another function block (PCO_MOT)

Tank
Level measurement

Drain

AC500 Program

- SP_EXT
- PV
- TRACK
- TRACK_SP
- BAL
- BAL_SP
- REVACT
- EN_AUTO
- ACT_POS
- RAMP_TIME
- RAMP_ACTIVE
- SCADA

OUT
OUT_MAX
OUT_MIN
ERR
ERNO

SCADA Interface
Setpoint

Workstation
Scaling of the SCADA Parameters

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to analog output of the AC500 I/Os.

If the setpoint and the process value do not have equal scaling, there is a need for scaling the SCADA.KP_PAR parameter to achieve the right proportional gain.

Scaling of the Parameter SCADA.KP_PAR

The scaling of the proportional coefficient (SCADA.KP_PAR) is dependant on the scaling of the process value (PV).

The PV typically comes from the function block PCO_ANAALM.

The process value, e.g. 0 bar ... 16 bar is rescaled to the value 0 ... 16000 (SCALE_MIN - SCALE_MAX).

The scaling of the KP [%] (RAWMAX in SCADA) is then calculated by the formula:

\[
KP \% = \frac{[PI \text{ out max} - PI \text{ out min}]}{[PV \text{ max} - PV \text{ min}]} \times 100 \%
\]

If the wanted range of SCADA.KP_PAR in SCADA (RNGMAX) is chosen to be 100 %, the RAWMAX in SCADA is calculated to be:

\[
RAWMAX \text{ in SCADA} = \frac{[27648 - 0]}{[16000 - 0]} \times 100 \% = 345
\]

RNGMAX = SCADA.Signal Configuration.Range.High limit

RAWMAX = SCADA.Signal Configuration.Range.High limit in PLC
Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.

Input Description

**SP_EXT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

External setpoint.
Calculated setpoint or output from another PID controller (e.g. PCO_PIDCON) in a cascade control system.

**PV**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

Process value (PV) to be controlled.

**TRACK**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking mode.

PCO_PIDCON can be set in Tracking mode if the value of TRACK is TRUE.
Output of the controller will be set to TRACK_VALUE.
Tracking mode has higher priority compared to balancing mode.
TRACK is independent of operation mode of PCO_PIDCON.

**TRACK_SP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking setpoint.
Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.
**BAL**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing mode.

PCO_PIDCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.

Output of the controller will be set to BAL_SP.

BAL is independent of operation mode of PCO_PIDCON.

**BAL_SP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing setpoint.

Output of the controller will be set to BAL_SP if value of BAL is TRUE.

**REVACT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Reverse Action

Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.

Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

**EN_AUTO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable automatic mode of the PCO_PIDCON.

PCO_PIDCON can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_PIDCON is forced in manual mode if value of EN_AUTO is FALSE.

**ACT_POS**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback speed of motor.

**RAMP_TIME**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#30s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Actuator ramp time from 0 % ... 100 %

**RAMP_ACTIVE**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.

The value RAMP_ACTIVE is FALSE activates no ramp.
Output Description

**OUT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

Controller output to the motor to attain the desired speed of the motor.

**OUT_MAX**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Output max limit reached, controller at max limit.

**OUT_MIN**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Output min limit reached, controller at min limit.

**ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, Input Parameter error.

**ERNO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number
Output provides an error identifier if an invalid value was applied to an input.
ERNO always must be considered together with the output ERR.
The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”.
### Input/Output Description

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Default value</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDUP5</td>
<td>BOOL</td>
<td></td>
<td>Manual command to increase output by 5 % part of the PCO_PIDCON</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDUP1</td>
<td>BOOL</td>
<td></td>
<td>Manual command to increase output by 1 % part of the PCO_PIDCON</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW1</td>
<td>BOOL</td>
<td></td>
<td>Manual command to decrease output by 1 % part of the PCO_PIDCON</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW5</td>
<td>BOOL</td>
<td></td>
<td>Manual command to decrease output by 5 % part of the PCO_PIDCON</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CM_CMDON</td>
<td>BOOL</td>
<td></td>
<td>Manual command for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
<td></td>
<td>Auto command to set the internal setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
<td></td>
<td>Auto command to set the external setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CAUTO</td>
<td>BOOL</td>
<td></td>
<td>Controller of the PCO_PIDCON is in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPIACT</td>
<td>BOOL</td>
<td></td>
<td>Internal setpoint of the PCO_PIDCON is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPXACT</td>
<td>BOOL</td>
<td></td>
<td>External setpoint of the PCO_PIDCON is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CTACT</td>
<td>BOOL</td>
<td></td>
<td>Tracking setpoint of the PCO_PIDCON is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_H</td>
<td>BOOL</td>
<td></td>
<td>Process value at high limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_L</td>
<td>BOOL</td>
<td></td>
<td>Process value at low limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_ERR</td>
<td>BOOL</td>
<td></td>
<td>Process value error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OUT_PAR</td>
<td>INT</td>
<td></td>
<td>Manual position setpoint</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
<td></td>
<td>Internal setpoint value</td>
<td>Input</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
<td>Proportional coefficient (gain) % of the controller</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
<td>Time constant for D-part of the controller</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
<td>Time constant for integration of the controller</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
<td>Output high limit (0 ... 10000) of the controller</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
<td>Output low limit (0 ... 10000) of the controller</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>SCADA.ACT_VALUE</td>
<td>INT</td>
<td>Actual speed of the motor, scaling as input ACT_POS (0 ... 10000)</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>SCADA.OUT_VALUE</td>
<td>INT</td>
<td>Output from the controller Scale 0 ... 10000</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>SCADA.PV_VALUE</td>
<td>INT</td>
<td>Process value to be controlled</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>SCADA.SPX_VALUE</td>
<td>INT</td>
<td>External setpoint value</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>SCADA.OSMsg1 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_PIDCON</td>
<td>Output</td>
<td></td>
</tr>
</tbody>
</table>

*) structure described separately

### SCADA.OSMsg1

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_PIDCON.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Internal setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 3</td>
<td>External setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Tracking setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 7</td>
<td>PID controller in manual mode</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 10</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 11</td>
<td>PID controller in automatic mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 13</td>
<td>PID controller not released for automatic mode</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
This function block is designed for controlling a variable position valve. The function block is similar to PCO_VALV (open/close valve), but in addition to PCO_VALV the function block has a built in PID controller, used for position control of the valve.

The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.

- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.

- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.

To make the controller follows various setpoints, the following variables need to be set:

*The values set below are default values, the user can change based on the application.*
### Settings in SCADA Side

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>When the controller must follow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Internal Setpoint</td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
<td>1</td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
<td>10000</td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
<td>0</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
<td>Needs to be set</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
<td>-</td>
</tr>
</tbody>
</table>

### Settings in FB Side

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>When the controller must follow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Internal Setpoint</td>
</tr>
<tr>
<td>EN_AUTO</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>SP_EXT</td>
<td>INT</td>
<td>-</td>
</tr>
<tr>
<td>TRACK</td>
<td>BOOL</td>
<td>-</td>
</tr>
<tr>
<td>TRACK_SP</td>
<td>INT</td>
<td>-</td>
</tr>
<tr>
<td>EN_ACC</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>EN_DEC</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.

The manual positioning can be done in two ways, both controlled form the SCADA faceplate.

1. **Stepwise 1 % up / down or 5 % up / down.**
   
   This is manually set by the following variables on the SCADA faceplate:
   - SCADA.CMDUP1
   - SCADA.CMDUP5
   - SCADA.CMDDW1
   - SCADA.CMDDW5

   The controller output varies based on the command given.

2. **Download a new position.** Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.
If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

---

Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

---

6 different alarms can be connected to the function block (ALARM_1 → ALARM_6, Torque switch etc.).

The valve can be switched to automatic or manual mode from the SCADA faceplate, if the input EN_AUTO = TRUE.

The position of the valve is controlled by the PID controller based on the process values.

If EN_Auto = FALSE, then the function block is forced to manual mode.

The position of the valve is controlled by the OUT_PAR value or by the commands CMDDW1, CMDDW5, CMDUP5, CMDUP1.

In automatic mode the valve is position controlled using the SP_EXT input as the position reference.

In manual mode the valve is position controlled using a setpoint from the SCADA interface, OUT_PAR.

Using the function block inputs EN_OPN and EN_CLS it is possible to prevent the valve from opening or closing by setting the respective input = FALSE. The inputs OPN and CLS are feedback indicators of the valve open and valve closed position. This is only used as display information for the SCADA faceplate.

Function error is generated if the controller is in automatic mode and then force to manual mode.

Function error will be active for 3 seconds. Controller will be force to manual mode if READY = TRUE or EN_AUTO = FALSE.

This function block can be used in combination with the object type VALVCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

---

Examples

The function block is used for controlling variable position valve which in turn controls the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Valve
- Main power supply, circuit breaker (or similar)

Manual mode:

- Valve controlled by the operator.

Automatic mode:

- Valve controlled by the AC500 controller.
PCO_VALVCON
Manual Mode
Example

Tank
Level measurement
Drain

Position controlled valve
Position Feedback
Position Setpoint

Main power supply of the valve OK
Toque fault from the valve

AC500 Program
SP_EXT
PV
TRACK
TRACK_SP
BAL
BAL_SP
REVACT
EN_AUTO
EN_OPN
EN_CLS
ACT_POS
OPN
CLS
RDY
REMOTE
INHIB_ERR
ALARM_1
ALARM_2
ALARM_3
ALARM_4
ALARM_5
ALARM_6
RAMP_TIME
RAMP_ACTIVE
SCADA

SCADA Interface
Setpoint

Workstation
Scaling of the SCADA Parameters

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to analog output of the AC500 I/Os.

If the setpoint and the process value do not have equal scaling, there is a need for scaling the SCADA.KP_PAR parameter to achieve the right proportional gain.
Scaling of the Parameter SCADA.KP_PAR

The scaling of the proportional coefficient (SCADA.KP_PAR) is dependant on the scaling of the process value (PV).

The PV typically comes from the function block PCO_ANAALM.

The process value, e.g. 0 bar ... 16 bar is rescaled to the value 0 ... 16000 (SCALE_MIN - SCALE_MAX).

The scaling of the KP [%] (RAWMAX in SCADA) is then calculated by the formula:

$$\text{KP [%]} = \frac{\text{PI out max} - \text{PI out min}}{\text{PV max} - \text{PV min}} \times 100\%$$

If the wanted range of SCADA.KP_PAR in SCADA (RNGMAX) is chosen to be 100 %, the RAWMAX in SCADA is calculated to be:

- RAWMAX in SCADA = \([27648 - 0] / [16000 - 0] \times 100 \% = 345\]

RNGMAX = SCADA.Signal Configuration.Range.High limit

RAWMAX = SCADA.Signal Configuration.Range.High limit in PLC

Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.

Input Description

![Input Description Image]
### SP_EXT

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

External setpoint.
Calculated setpoint or output from another PID controller (e.g. PCO_VALVCON) in a cascade control system.

### PV

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 27648</td>
<td>-</td>
</tr>
</tbody>
</table>

Process value (PV) to be controlled.

### TRACK

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking mode.
PCO_VALVCON can be set in Tracking mode if the value of TRACK is TRUE.
Output of the controller will be set to TRACK_VALUE.
Tracking mode has higher priority compared to balancing mode.
TRACK is independent of operation mode of PCO_VALVCON.

### TRACK_SP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Tracking setpoint.
Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.

### BAL

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing mode.
PCO_VALVCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.
Output of the controller will be set to BAL_SP.
BAL is independent of operation mode of PCO_VALVCON.

### BAL_SP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

Balancing setpoint.
Output of the controller will be set to BAL_SP if value of BAL is TRUE.
### REVACT

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Reverse Action**

Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.

Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

### EN_AUTO

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Enable automatic mode of the PCO_VALVCON.**

PCO_VALVCON can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_VALVCON is forced in manual mode if value of EN_AUTO is FALSE.

### EN_OPN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Enable valve to open.**

Valve can be opened if value of EN_OPN is TRUE.

Valve cannot be opened if value of EN_OPN is FALSE.

### EN_CLS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Enable valve to close.**

Valve can be closed if value of EN_CLS is TRUE.

Valve cannot be closed if value of EN_CLS is FALSE.

### ACT_POS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>0 ... 10000</td>
<td>-</td>
</tr>
</tbody>
</table>

**Position feedback of valve.**

### OPN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Feedback signal from the valve.**

If the value of OPN is TRUE, the valve is open.

### CLS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

**Feedback signal from the valve.**

If the value of CLS is TRUE, the valve is closed.
**RDY**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Valve is ready for operation.
Valve is ready for operation if value of RDY is TRUE.
Value of RDY is FALSE results in an external error.

**REMOTE**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Allow valve to be controlled from SCADA.
Value of REMOTE is TRUE control from SCADA and function block is enabled.
Value of REMOTE is FALSE valve is controlled only from function block.

**INHIB_ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of alarms.
If value of INHIB_ERR is TRUE all alarms from PCO_VALVCON are suppressed.

**Alarm_1**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 1.
If value of ALARM_1 is TRUE Alarm_1 is active.
Independent of operation mode of PCO_VALVCON.

**Alarm_2**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 2.
If value of ALARM_2 is TRUE Alarm_2 is active.
Independent of operation mode of PCO_VALVCON.

**Alarm_3**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 3.
If value of ALARM_3 is TRUE Alarm_3 is active.
Independent of operation mode of PCO_VALVCON.
### Alarm_4

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_VALVCON.

### Alarm_5

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_VALVCON.

### Alarm_6

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_VALVCON.

### RAMP_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#30s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Actuator ramp time from 0 % ... 100 %

### RAMP_ACTIVE

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.

The value RAMP_ACTIVE is FALSE activates no ramp.
Output Description

Controller output to the valve to attain the desired position of the valve.

<table>
<thead>
<tr>
<th>OUT_MAX</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Output max limit reached, controller at max limit.

<table>
<thead>
<tr>
<th>OUT_MIN</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Output min limit reached, controller at min limit.
Common alarm, including:

- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6

Error number

Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Input/Output Description

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.OPEN</td>
<td>BOOL</td>
<td>Valve is open</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CLOSED</td>
<td>BOOL</td>
<td>Valve is closed</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.REMOTE</td>
<td>BOOL</td>
<td>Valve can be controlled from OS</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REMOTE = FALSE → Local operation</td>
<td></td>
</tr>
<tr>
<td>SCADA.READY</td>
<td>BOOL</td>
<td>Valve is ready for operation</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.FNCERR</td>
<td>BOOL</td>
<td>Functional error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.EXTERR</td>
<td>BOOL</td>
<td>External error</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(is generated when the valve is not ready)</td>
<td></td>
</tr>
<tr>
<td>SCADA.AL1</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 1</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL2</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 2</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL3</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 3</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL4</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 4</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL5</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 5</td>
<td>Output</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.AL6</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 6</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.ACT_VALUE</td>
<td>INT</td>
<td>Actual position of the valve, scaling as input ACT_POS (0 ... 10000)</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CM_CMDON</td>
<td>BOOL</td>
<td>Manual command for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CI_CMDON</td>
<td>BOOL</td>
<td>Auto command to set the internal setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CX_CMDON</td>
<td>BOOL</td>
<td>Auto command to set the external setpoint for the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDUP5</td>
<td>BOOL</td>
<td>Manual command to increase output by 5 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDUP1</td>
<td>BOOL</td>
<td>Manual command to increase output by 1 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW1</td>
<td>BOOL</td>
<td>Manual command to decrease output by 1 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDDW5</td>
<td>BOOL</td>
<td>Manual command to decrease output by 5 %</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CAUTO</td>
<td>BOOL</td>
<td>Controller is in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPIACT</td>
<td>BOOL</td>
<td>Internal setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPXACT</td>
<td>BOOL</td>
<td>External setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CTACT</td>
<td>BOOL</td>
<td>Tracking setpoint is active</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_H</td>
<td>BOOL</td>
<td>Process value at high limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_L</td>
<td>BOOL</td>
<td>Process value at low limit</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PV_ERR</td>
<td>BOOL</td>
<td>Process value error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OUT_PAR</td>
<td>INT</td>
<td>Manual position setpoint</td>
<td>Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scale 0 ... 10000</td>
<td></td>
</tr>
<tr>
<td>SCADA.SPI_PAR</td>
<td>INT</td>
<td>Internal setpoint value</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.KP_PAR</td>
<td>INT</td>
<td>Proportional coefficient (gain) % of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.TD_PAR</td>
<td>INT</td>
<td>Time constant for D-part of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.TI_PAR</td>
<td>INT</td>
<td>Time constant for integration of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OH_PAR</td>
<td>INT</td>
<td>Output high limit (0 ... 10000) of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OL_PAR</td>
<td>INT</td>
<td>Output low limit (0 ... 10000) of the controller</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.OUT_VALUE</td>
<td>INT</td>
<td>Output from the controller</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scale 0 ... 10000</td>
<td></td>
</tr>
<tr>
<td>SCADA.PV_VALUE</td>
<td>INT</td>
<td>Process value to be controlled</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SPX_VALUE</td>
<td>INT</td>
<td>External setpoint value</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg1 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_VALVCON</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg2 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_VALVCON</td>
<td>Output</td>
</tr>
</tbody>
</table>

*) structure described separately
**SCADA.OSMsg1**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_VALVCON.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Valve is open</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Valve is closed</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 4</td>
<td>External error (Not ready)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Functional error</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Valve released for opening</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Valve released for closing</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Local operation (Not remote)</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Common alarm (External alarm + Functional alarm + ALARM_1 ... ALARM_6)</td>
</tr>
<tr>
<td>Bit 10</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 11</td>
<td>Motor in automatic mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Motor not released for automatic mode</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Motor released for increase speed</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Motor released for decrease speed</td>
</tr>
<tr>
<td>Data type</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>WORD</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_VALVCON.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Internal setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 3</td>
<td>External setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Track setpoint (PID controller)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Auxiliary alarm 1</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Auxiliary alarm 2</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Auxiliary alarm 3</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Auxiliary alarm 4</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Auxiliary alarm 5</td>
</tr>
<tr>
<td>Bit 10</td>
<td>PID controller in automatic mode</td>
</tr>
<tr>
<td>Bit 11</td>
<td>PID controller in manual mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Auxiliary alarm 6</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
1.2.2.1.2 Indications

PCO_ALARM

The function block is used to send an alarm to the SCADA system.

The alarm sent to SCADA is extended minimum 3 seconds to ensure alarm detection in the SCADA system, even if the alarm at the input goes off before 3 seconds. The variable that represents the alarm in the SCADA side is "Status". This function block can be used in combination with the object type ALARM_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input Description

<table>
<thead>
<tr>
<th>IN</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Alarm input.

If value of IN is TRUE then an alarm is sent to SCADA.

INHIB_ALARM

<table>
<thead>
<tr>
<th>INHIB_ALARM</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_ALARM</td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Alarm inhibit.

If value of INHIB_ALARM is TRUE then no alarm is sent to SCADA regardless of the value on IN.

Output Description

<table>
<thead>
<tr>
<th>SCADA</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA</td>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA_STATUS</td>
<td>BOOL</td>
<td>Alarm status, TRUE = Alarm</td>
<td>Output</td>
</tr>
</tbody>
</table>
PCO_BIN

The function block is used to send an event or an indication to the SCADA system. The event is entered at IN input of the function block and its presence is depicted as "Status" variable in SCADA. The event sent to SCADA is extended minimum 3 seconds to ensure detection in the SCADA system. This function block can be used in combination with the object type BIN_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input Description

**IN**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Event input.
If value of IN is TRUE then an event is sent to SCADA.

Output Description

**SCADA**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.STATUS</td>
<td>BOOL</td>
<td>Event status. If value is TRUE then event is depicted, else no event.</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The event is maintained for a minimum of 3 seconds, even if the event becomes FALSE before that.</td>
<td></td>
</tr>
</tbody>
</table>
1.2.2.1.3 Measurements

PCO_ANA

The function block is used to send an analogue value to the SCADA system with hysteresis limits.

The hysteresis limit can be entered from the FB. The value of the analog signal is depicted in SCADA through a variable named value. SCADA.VALUE is only updated, if IN input (analog input) has changed more than the HYS input limits defined.

For example:

Let the analog input be defined as 2, the SCADA.VALUE gets assigned as 2. Suppose the hysteresis is 3 then the range of up to which the input can vary is (2-3) and (2+3) which is -1 ... 5.

If an input -1 is given, then the SCADA.VALUE does not change (as it is in the range of -1 ... 5).

If -2 is given at the input, then the SCADA.VALUE gets set to -2 and then the input range becomes -5 ... 1.

There is no scaling in the function block.

This function block can be used in combination with the object type ANA_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input Description

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Analogue input that must be sent to the SCADA system.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Dead band limit for input.

Output Description
### SCADA

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication to and from SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.VALUE</td>
<td>INT</td>
<td>Analog input within the hysteresis limits.</td>
<td>Output</td>
</tr>
</tbody>
</table>
PCO_ANAALM

The function block is designed for controlling an analogue input with limit supervision. The function block can be configured to handle the different ranges of the analogue input, due to different electrical signals. E.g.

- 4 mA ... 20 mA
- 0 V ... 10 V or
- PT100 /PT1000

Please refer to AC500 hardware manual for detailed information regarding analogue input modules and different parameters.

This function block checks whether the analog output of the FB are within HH, H or LL, L limits. The limits need to be entered from the SCADA side.

- SCADA.LIMH2
- SCADA.LIMH1
- SCADA.LIML1
- SCADA.LIML2

The function block rescales the analog input (IN) from the actual value to SCALE_MIN → SCALE_MAX.

If TEMP_PT100_PT1000 or PB are to be set = TRUE, no rescaling of IN is made.

If an analogue input is connected to an I/O module and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

Zero point suppression. The function block will force the input to zero when the input is between 0 and 0 - IN_HYS

The value can be preset to SCALE_MIN (FORCE_SCALE_MIN = TRUE) or SCALE_MAX (FORCE_SCALE_MAX = TRUE).

The output value from the function block can be frozen to a specific value (FREEZE = TRUE and the specific value on FREEZE_SP).
The limit supervision consists of up to two high and two low alarms.

A SENSOR ERROR is generated in the SCADA side in the following conditions:
1. When the analog input is not within the tolerance of 1000, hence the input can vary over a range of -1000 ... 28648, after which it generates an error.
2. The output obtained at the end is not within the limits of SCALE_MIN and SCALE_MAX.
3. When the PB input is activated and the PB_STATUS is not equal to 128.

_Inputs and outputs can stay unconnected, if their functions are not needed._

This function block can be used in combination with the object type ANAALM_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input Description

<table>
<thead>
<tr>
<th>LIM_HYS</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REAL</td>
<td>0.5</td>
<td>0 ... 100</td>
<td>%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dead band for Alarm Limit. % of scale (SCALE_MAX – SCALE_MIN).

<table>
<thead>
<tr>
<th>LIM_HH_INHIB</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Suppression of highhigh alarm.
If value of LIM_HH_INHIB is TRUE, then suppress highhigh alarm.
### LIM_H_INHIB

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of high alarm.
If value of LIM_H_INHIB is TRUE, then suppress high alarm.

### LIM_L_INHIB

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of low alarm.
If value of LIM_L_INHIB is TRUE, then suppress low alarm.

### LIM_LL_INHIB

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of lowlow alarm.
If value of LIM_LL_INHIB is TRUE, then suppress lowlow alarm.

### IN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Analogue input.
Value to SCADA is scaled according to SCALE_MIN and SCALE_MAX.
\( 0 \ldots 27648 \rightarrow SCALE\_MIN \cdot SCALE\_MAX. \)

### SCALE_MAX

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>27648</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Scaling parameter. Maximum value for output to SCADA and output of the function block.
IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

### SCALE_MIN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Scaling parameter. Minimum value for output to SCADA and output of the function block.
IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

### IN_HYS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>0.5</td>
<td>0 ... 100</td>
<td>%</td>
</tr>
</tbody>
</table>

Dead band for analogue input.
If \( \text{IN} + \text{IN\_HYS} > \text{SCADA} \) or \( \text{IN} - \text{IN\_HYS} < \text{SCADA} \), then SCADA value is updated.
### EN_LIVE_ZERO

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable live zero for analogue input.

An analogue input is connected to an I/O module or CPU and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVEZERO is TRUE.

### INHIB_ERR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of all alarms.

If value of INHIB_ERR is TRUE all alarms are suppressed.

### FORCE_SCALE_MIN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force output from function block.

If value of FORCE_SCALE_MIN is TRUE, output for SCADA and OUT are forced to SCALE_MIN.

When FORCE_SCALE_MAX and FORCE_SCALE_MIN inputs are made high at the same time, FORCE_SCALE_MAX has higher priority.

### FORCE_SCALE_MAX

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force output from function block.

If value of FORCE_SCALE_MAX is TRUE, output for SCADA and OUT are forced to SCALE_MAX.

When FORCE_SCALE_MAX and FORCE_SCALE_MIN inputs are made high at the same time, FORCE_SCALE_MAX has higher priority.

### FREEZE

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force output from function block.

If value of FREEZE is TRUE, output for SCADA and OUT are forced to FREEZE_SP.

### FREEZE_SP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Freeze setpoint.

If value of FREEZE is TRUE, the SCADA and OUT will be forced to FREEZE_SP.
PB

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Select PROFIBUS type of input.

If value of PB is TRUE, the analogue input is not from an I/O module but from a PROFIBUS communication line.

No scaling of the SCADA and OUT value.

PB_STATUS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>128</td>
<td>0 ... 255</td>
<td>-</td>
</tr>
</tbody>
</table>

Status of PROFIBUS communication.

If value of PB_STATUS is not equal to 128, a sensor error is generated.

If value of PB_STATUS is 128, measurement is ok.

FIL_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#2s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Filter time. 1st order filter for the input damps the variation of input.

Output Description

Scaled output, output scaled according to SCALE_MIN and SCALE_MAX.
**ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, Input Parameter error.

**ERNO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number

Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”.

**Input/Output Description**

**SCADA**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.VALUE</td>
<td>INT</td>
<td>Rescaled input value.</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SENSE_ERR</td>
<td>BOOL</td>
<td>Sensor error. Open circuit or short circuit.</td>
<td>Output</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.H2</td>
<td>BOOL</td>
<td>High-high alarm (H2).</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.H1</td>
<td>BOOL</td>
<td>High alarm (H1).</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.L1</td>
<td>BOOL</td>
<td>Low alarm (L1).</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.L2</td>
<td>BOOL</td>
<td>Low-low alarm (L2).</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.LIMH2</td>
<td>INT</td>
<td>Parameter from OS: Limit for H2 alarm.</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.LIMH1</td>
<td>INT</td>
<td>Parameter from OS: Limit for H1 alarm.</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.LIML1</td>
<td>INT</td>
<td>Parameter from OS: Limit for L1 alarm.</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.LIML2</td>
<td>INT</td>
<td>Parameter from OS: Limit for L2 alarm.</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.H2_VALUE</td>
<td>INT</td>
<td>Parameter to SCADA: Limit for H2 alarm</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.H1_VALUE</td>
<td>INT</td>
<td>Parameter to SCADA: Limit for H1 alarm</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.L1_VALUE</td>
<td>INT</td>
<td>Parameter to SCADA: Limit for L1 alarm</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.L2_VALUE</td>
<td>INT</td>
<td>Parameter to SCADA: Limit for L2 alarm</td>
<td>Output</td>
</tr>
</tbody>
</table>
The function block is designed for controlling an analogue input without limit supervision.

The function block can be configured to handle the different ranges of the analogue input, due to different electrical signals. E.g.

- 4 mA ... 20 mA
- 0 V ... 10 V or
- PT100 / PT1000

Please refer to AC500 hardware manual for detailed information regarding analogue input modules and different parameters.

The function block rescales the analog input (IN) from the actual value to SCALE_MIN \( \rightarrow \) SCALE_MAX.

If TEMP_PT100_PT1000 or PB are to be set = TRUE, no rescaling of IN is made.

If an analogue input is connected to an I/O module and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

Zero point suppression. The function block will force the input to zero when the input is between 0 and 0 - IN_HYS

A SENSOR ERROR is generated in the SCADA side in the following conditions:

1. When the analog input is not within the tolerance of 1000, hence the input can vary over a range of -1000 \( \ldots \) 28648, after which it generates an error.
2. The output obtained at the end is not within the limits of SCALE_MIN and SCALE_MAX.
3. When the PB input is activated and the PB_STATUS is not equal to 128.

Inputs and outputs can stay unconnected, if their functions are not needed.

This function block can be used in combination with the object type ANALIM_PLCL included in PLC Object Library (an object library for 800xA based PLC Connect).
### Input Description

![PCO_ANALIM](image)

#### IN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Analogue input. Value to SCADA is scaled according to SCALE_MIN and SCALE_MAX.  
0 ... 27648 $\rightarrow$ SCALE_MIN - SCALE_MAX.

#### SCALE_MAX

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>27648</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Scaling parameter. Maximum value for output to SCADA and output of the function block.  
IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

#### SCALE_MIN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Scaling parameter. Minimum value for output to SCADA and output of the function block.  
IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

#### IN_HYS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>0.5</td>
<td>0 ... 100</td>
<td>%</td>
</tr>
</tbody>
</table>

Dead band for analogue input.  
If IN + IN_HYS > SCADA or IN - IN_HYS < SCADA, then SCADA value is updated.

#### EN_LIVE_ZERO

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable live zero for analogue input.  
An analogue input is connected to an I/O module or CPU and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.
**TEMP_PT100_PT1000**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Select temperature sensor type of input.

If value of TEMP_PT100_PT1000 is TRUE, a temperature sensor is connected direct on an I/O module.

No scaling of the SCADA and OUT value.

**PB**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Select PROFIBUS type of input.

If value of PB is TRUE, the analogue input is not from an I/O module but from a PROFIBUS communication line.

No scaling of the SCADA and OUT value.

**PB_STATUS**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>128</td>
<td>0 ... 255</td>
<td>-</td>
</tr>
</tbody>
</table>

Status of PROFIBUS communication.

If value of PB_STATUS is not equal to 128, a sensor error is generated.

If value of PB_STATUS is 128, measurement is ok.

**FIL_TIME**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#2s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Filter time. 1st order filter for the input damps the variation of input.

**Output Description**

![Diagram](PCO_ANALIM)

**OUT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Scaled output, output scaled according to SCALE_MIN and SCALE_MAX.
**ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, Input Parameter error.

**ERNO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number

Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”.

**SCADA**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication to and from SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.W-L-UIE</td>
<td>INT</td>
<td>Rescaled input value.</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.SENSE_ERR</td>
<td>BOOL</td>
<td>Sensor error. Open circuit or short circuit.</td>
<td>Output</td>
</tr>
</tbody>
</table>
1.2.2.1.4 Motor

PCO_MOT

This function block is designed for controlling a fixed speed motor.

Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_START and CMD_STOP can be configured. E.g. if CMD_START and CMD_STOP should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.)

The motor can be switched to automatic mode from the SCADA faceplate if the input EN_AUTO is TRUE.

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Set Motor to Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN_AUTO</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>REMOTE</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Set Motor to Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Rising edge needs to be given.</td>
</tr>
</tbody>
</table>
**Start / Stop Motor in Auto Mode**

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Start / Stop Motor in Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
<td>The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.</td>
</tr>
<tr>
<td>EN_START</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>EN_STOP</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>AUTO_CMD_START</td>
<td>BOOL</td>
<td>Rising edge needs to be given to AUTO_CMD.START input of FB</td>
</tr>
<tr>
<td>AUTO_CMD_STOP</td>
<td>BOOL</td>
<td>Rising edge needs to be given to AUTO_CMD_STOP input of FB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Start / Stop Motor in Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>No settings needs to be done</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Set Motor to Manual Mode**

The motor can be switched to manual mode.

If the input EN_AUTO is FALSE then the function block is forced to manual mode.

The motor can be operated in manual mode if EN_AUTO is either TRUE or FALSE.

The difference is that if EN_AUTO is TRUE, the switching between Auto and Manual is possible, else the motor and controller is both to manual.

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Set Motor to Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Set Motor to Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
<td>Rising edge of SCADA.CMDMAN input to reset auto mode if it was set before</td>
</tr>
</tbody>
</table>

**Start / Stop Motor in Manual Mode**

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Start / Stop Motor in Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
<td>The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.</td>
</tr>
<tr>
<td>EN_START</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>EN_STOP</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Start / Stop Motor in Manual Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMDSTR</td>
<td>BOOL</td>
<td>Rising edge of CMDSTR needs to be given.</td>
</tr>
<tr>
<td>CMDSTP</td>
<td>BOOL</td>
<td>Rising edge of CMDSTP needs to be given.</td>
</tr>
</tbody>
</table>
In automatic mode the motor can be started/stopped using the function block inputs (AUTO_CMD_START/AUTO_CMD_STOP).

In manual mode the motor can be started and stopped from the SCADA faceplate.

The function block includes supervision of the feedback signals RUN and STOP from the motor. If the motor fails to run or stop within the supervision time INHIB_FCT_ERR_TIME, it will generate a functional error.

An external error is generated when READY input is false.

This function block can be used in combination with the object type MOT_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

**Examples**

The function block is used for controlling a fixed speed motor, which can be used for controlling the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level switch (high and low)
- Pump
- Main power supply, circuit breaker (or similar)

**Manual mode:**

- Pump controlled by the operator.

**Automatic mode:**

- Pump controlled by the AC500 controller.
PCO_MOT
Manual Mode
Example
PCO_MOT
Automatic Mode
Example

On/Off controlled pump

Feedback Running
Feedback Stopped
Stop Command
Start Command

AC500 Program

PCO_MOT
CMD_START
CMD_STOP
ERNO

<
AUTO_CMD_START
AUTO_CMD_STOP
EN.AUTO
EN.Stop
EMCY_START
EMCY_STOP
RUN
STOP
RDY
REMOTE
INHIB_ERR
EXT_ACK
INHIB_FCT_ERR_TIME
ALARM_1
ALARM_2
ALARM_3
ALARM_4
ALARM_5
ALARM_6
BEHAVE_ON_ERROR
OUTPUT_HANDLING
SCADA

Main power supply of the pump OK
Termister fault from the pump motor

SCADA is needed to set the function block to auto mode

Tank
Level measurement
Drain

Workstation
**Input Description**

<table>
<thead>
<tr>
<th>AUTO_CMD_START</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Force the motor to start if it is in automatic mode.

E.g. AUTO_CMD_START could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

<table>
<thead>
<tr>
<th>AUTO_CMD_STOP</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Force the motor to stop if the actuator is in automatic.

AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.
**EN_AUTO**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable automatic mode of the PCO_MOT.  
PCO_MOT can be set in automatic mode if value of EN_AUTO is TRUE.  
PCO_MOT is forced in manual mode if value of EN_AUTO is FALSE.  
But the motor can also be operated in manual mode if EN_AUTO is TRUE provided that manual start and stop selection is made on the SCADA side.

**EN_START**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable start of the motor.  
Motor can be started if value of EN_START is TRUE.  
Motor cannot be started if value of EN_START is FALSE.  
EN_START is independent of operation mode of PCO_MOT.  
EN_START has no effect in case of emergency start of function block motor EMCY_START.  
EN_START has no effect on a motor already running.

**EN_STOP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Enable stop of the motor.  
Motor can be stopped if value of EN_STOP is TRUE.  
Motor cannot be stopped if value of EN_STOP is FALSE.  
EN_STOP is independent of operation mode of PCO_MOT.  
EN_STOP has no effect in case of emergency stop of function block motor EMCY_STOP.  
EN_STOP has no effect on an already stopped motor.

**EMCY_START**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Emergency start of motor.  
If value of EMCY_START is TRUE and value of EMCY_STOP is FALSE, the motor will start.  
If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.  
Independent of operation mode of PCO_MOT.

**EMCY_STOP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Emergency stop of the motor.  
If value of EMCY_STOP is TRUE, the motor will stop.  
If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.  
Independent of operation mode of PCO_MOT.
### RUN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback signal from the motor.
Motor is running if the value of RUN is TRUE.
Independent of operation mode of PCO_MOT.

### STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Feedback signal from the motor.
Motor is stopped if the value of STOP is TRUE.
Independent of operation mode of PCO_MOT.

### RDY

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Motor is ready for operation.
Motor is ready for operation if value of RDY is TRUE.
Value of RDY is FALSE results in an external error.
Independent of operation mode of PCO_MOT.

### REMOTE

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Allow motor to be controlled from SCADA.
Value of REMOTE is TRUE control from SCADA and function block is enabled.
Value of REMOTE is FALSE motor is controlled only from function block.
PCO_MOT will align the block according to feedback signals.
Independent of operation mode of PCO_MOT.

### INHIB_ERR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of alarms.
If value of INHIB_ERR is TRUE all alarms from PCO_MOT are suppressed.
Independent of operation mode of PCO_MOT.

### EXT_ACK

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

External acknowledge of alarms.
If value of EXT_ACK is TRUE all alarms from PCO_MOTCON are acknowledged.
Independent of operation mode of PCO_MOT.
### INHIB_FCT_ERR_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#5s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Maximum delay time from command to response from process. If response is not received within this time limit, a function error will be generated.

### Alarm_1

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 1. If value of ALARM_1 is TRUE Alarm_1 is active. Independent of operation mode of PCO_MOT.

### Alarm_2

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 2. If value of ALARM_2 is TRUE Alarm_2 is active. Independent of operation mode of PCO_MOT.

### Alarm_3

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 3. If value of ALARM_3 is TRUE Alarm_3 is active. Independent of operation mode of PCO_MOT.

### Alarm_4

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 4. If value of ALARM_4 is TRUE Alarm_4 is active. Independent of operation mode of PCO_MOT.

### Alarm_5

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 5. If value of ALARM_5 is TRUE Alarm_5 is active. Independent of operation mode of PCO_MOT.
### Alarm_6

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_MOT.

### BEHAVE_ON_ERROR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>0</td>
<td>0 ... 3</td>
<td>-</td>
</tr>
</tbody>
</table>

Actions that need to be executed by the FB at the event of error as set by the user.

The user can set a range of values from 0 ... 3.

- BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).
- BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.
- BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.
- BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto / Manual) of PCO_MOT.

### OUTPUT_HANDLING

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>1</td>
<td>0 ... 2</td>
<td>-</td>
</tr>
</tbody>
</table>

Behavior of command outputs, CMD_START / CMD_STOP.

The user can set a range of values from 0 ... 2.

- OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.
- OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP) to remain active at RUN or STOP feedback.
- With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_MOT.
Output Description

**CMD_START**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output start, to be connected to hardware output.

**CMD_STOP**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output stop, to be connected to hardware output.

**ERR**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, including:
- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_START
- EMCY_STOP
### ERNO

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number

Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in "Standard Function Block Libraries AC500" in "Error Messages of the Function Block Libraries".

### Input/Output Description

![Input/Output Diagram]

### SCADA

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDSTR</td>
<td>BOOL</td>
<td>Start command in Manual mode</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDSTP</td>
<td>BOOL</td>
<td>Stop command in Manual mode</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Auto command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
<td>Manual command</td>
<td>Input</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.CMDRES</td>
<td>BOOL</td>
<td>Acknowledge active alarms</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.AUTO</td>
<td>BOOL</td>
<td>Motor in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.IN</td>
<td>BOOL</td>
<td>Motor is running</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OUT</td>
<td>BOOL</td>
<td>Motor is stopped</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.REMOTE</td>
<td>BOOL</td>
<td>Motor can be controlled from SCADA REMOTE = FALSE → Local operation</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.READY</td>
<td>BOOL</td>
<td>Motor is ready for operation</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.FNCERR</td>
<td>BOOL</td>
<td>Functional error</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.EXTERR</td>
<td>BOOL</td>
<td>External error</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(is generated when the motor is not ready)</td>
<td></td>
</tr>
<tr>
<td>SCADA.AL1</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 1</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL2</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 2</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL3</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 3</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL4</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 4</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL5</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 5</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL6</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 6</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg1</td>
<td>WORD</td>
<td>Word representing the status of the PCO_MOT</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg2</td>
<td>WORD</td>
<td>Word representing the status of the PCO_MOT</td>
<td>Output</td>
</tr>
</tbody>
</table>

*) structure described separately
<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_MOT.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Motor is running</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Motor is stopped</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Motor is starting</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Motor is stopping</td>
</tr>
<tr>
<td>Bit 4</td>
<td>External error (Not ready)</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Functional error</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Motor released for start</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Motor released for stop</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Local operation (Not remote)</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Common alarm (External alarm + Functional alarm + EMCY_START/EMCY_STOP + ALARM_1 ... ALARM_6)</td>
</tr>
<tr>
<td>Bit 10</td>
<td>Unacknowledged alarm</td>
</tr>
<tr>
<td>Bit 11</td>
<td>Motor in automatic mode</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Motor not released for automatic mode</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
## SCADA.OSMsg2

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_MOT.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Emergency In</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Emergency Out</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Auxiliary alarm 1</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Auxiliary alarm 2</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Auxiliary alarm 3</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Auxiliary alarm 4</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Auxiliary alarm 5</td>
</tr>
<tr>
<td>Bit 7</td>
<td>Auxiliary alarm 6</td>
</tr>
<tr>
<td>Bit 8</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 9</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 10</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 11</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 12</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 13</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 14</td>
<td>Not used</td>
</tr>
<tr>
<td>Bit 15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
1.2.2.1.5 Setpoints

PCO_ANASET

This function block is designed for receiving an analog setpoint from the SCADA system. This value is entered at .PAR variable of the SCADA side. The actual analog setpoint is entered at IN input of the function block and it is represented by .VALUE variable in the SCADA side.

This function block can be used in combination with the object type ANASET_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

### Input Description

**IN**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Actual setpoint.
Parameter is sent to SCADA.

### Output Description

**OUT**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>0</td>
<td>-32768 ... 32767</td>
<td>-</td>
</tr>
</tbody>
</table>

Setpoint received from SCADA.
Parameter received from SCADA.
### SCADA

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.VALUE</td>
<td>INT</td>
<td>Actual setpoint</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.PAR</td>
<td>INT</td>
<td>Setpoint from SCADA</td>
<td>Input</td>
</tr>
</tbody>
</table>

### PCO_BINSET

This function block is designed for receiving single command from the SCADA system.

EN_PULSE input is used to reset command from SCADA. CMD variable from SCADA side is used to give command.

If value of EN_PULSE is TRUE, the command will only be active for one program cycle.

The CMD variable is connected to OUT (output) of function block.

This function block can be used in combination with the object type BINSET_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

### Input Description

<table>
<thead>
<tr>
<th>IN</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

**IN**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Actual status, that is the result of a command from SCADA side.

<table>
<thead>
<tr>
<th>EN_PULSE</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

**EN_PULSE**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Pulse output.

If value of EN_PULSE is TRUE, CMD variable would be reset and OUT will only be active for one program cycle.
Output Description

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Single command from SCADA.
If value of EN_PULSE is TRUE the single command will only be active for one program cycle.

Input/Output Description

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRUCTURE</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Structure variable for communication between AC500 and SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.STATUS</td>
<td>BOOL</td>
<td>Actual status, feedback</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CMD</td>
<td>BOOL</td>
<td>Single command from SCADA</td>
<td>Input</td>
</tr>
</tbody>
</table>
This function block is designed for controlling a motor controlled ON/OFF valve. Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_OPN, CMD_STOP and CMD_CLS can be configured. E.g. if CMD_OPN should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.)

There are two modes in which the valve can be operated, Auto mode or Manual mode.

In automatic mode the valve can be opened/stopped/closed using the function block inputs (AUTO_CMD_OPN/AUTO_CMD_STOP/AUTO_CMD_CLS).

### Set Valve to Auto Mode

The valve is set to auto mode when the following conditions are satisfied:

<table>
<thead>
<tr>
<th>Settings in FB side</th>
<th>Data type</th>
<th>Set Valve to Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN_AUTO</td>
<td>BOOL</td>
<td>TRUE</td>
</tr>
<tr>
<td>REMOTE</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings in SCADA side</th>
<th>Data type</th>
<th>Set Valve to Auto Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Rising edge needs to be given.</td>
</tr>
</tbody>
</table>
### Settings in FB side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>EN_OPN</td>
<td>BOOL</td>
</tr>
<tr>
<td>EN_CLS</td>
<td>BOOL</td>
</tr>
<tr>
<td>AUTO_CMD_OPN</td>
<td>BOOL</td>
</tr>
<tr>
<td>AUTO_CMD_STP</td>
<td>BOOL</td>
</tr>
<tr>
<td>AUTO_CMD_CLS</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Settings in SCADA side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Set Valve to Manual Mode

In manual mode the valve can be opened and closed from the SCADA faceplate. The valve is set to manual mode when the following conditions are satisfied:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE</td>
<td>BOOL</td>
</tr>
<tr>
<td>RDY</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Settings in SCADA side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Open / Close Valve in Manual Mode

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INHIB_FCT_ERR_TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>EN_OPN</td>
<td>BOOL</td>
</tr>
<tr>
<td>EN_CLS</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

### Settings in SCADA side

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMDOPN</td>
<td>BOOL</td>
</tr>
<tr>
<td>CMDCLS</td>
<td>BOOL</td>
</tr>
</tbody>
</table>
This function block can be used in combination with the object type VALVCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

The function block includes supervision of the feedback signals OPN and CLS from the valve. If the valve fails to open or close within the supervision time INHIB_FCT_ERR_TIME, it will raise a functional error. Actions in case of a functional error are controlled by the parameter BEHAVE_ON_ERROR.

Examples

The function block is used for controlling a valve which in turn controls the level of fluid in a tank. These components are included in the following example:

- Tank
- Level switch (high and low)
- Valve
- Main power supply, circuit breaker (or similar)

Manual mode:
- Valve controlled by the operator.

Automatic mode:
- Valve controlled by the AC500 controller.
PCO_VALV
Manual Mode
Example

Tank
Level measurement
Drain

Feedback Closed
Feedback Open
Open/Close controlled Valve
Close Command
Open Command

Main power supply
of the Valve OK

Torque fault from
the Valve

AC500 Program

PCO_VALV
AUTO_CMD_OPN
AUTO_CMD_CLS
AUTO_CMD_STOP
EN_AUTO
EN_OPN
EN_CLS
EMCY_OPN
EMCY_CLS
EMCY_STOP
OPN
CLS
RDY
REMOTE
INHIB_ERR
EXT_ACK
INHIB_FCT_ERR_TIME
ALARM_1
ALARM_2
ALARM_3
ALARM_4
ALARM_5
ALARM_6
BEHAVE_ON_ERROR
OUTPUT_HANDLING
SCADA

SCADA Interface
Open/Close commands for valve

Workstation
PCO_VALV
Automatic Mode
Example

Tank

Level measurement

Drain

Main power supply of the Valve OK

Torque fault from the Valve

Open/Close controlled Valve

Feedback Closed
Feedback Open

Close Command
Open Command

AC500 Program

PCO_VALV

AUTO_CMD_OPN
AUTO_CMD_CLS
AUTO_CMD_STOP
EN_AUTO
EN_OPN
EN_CLS
EMCY_OPN
EMCY_CLS
EMCY_STOP
OPN
CLS
RDY
REMOTE
INHIB_ERR
EXT_ACK
INHIB_FCT_ERR_TIME
ALARM_1
ALARM_2
ALARM_3
ALARM_4
ALARM_5
ALARM_6
BEHAVE_ON_ERROR
OUTPUT_HANDLING
SCADA

SCADA Interface
Monitoring only
It is also used to set the function block to auto mode by the SCADA input SCADA.CMDAUT

Workstation
Input Description

<table>
<thead>
<tr>
<th>AUTO_CMD_OPN</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force the valve to open, if the valve is in automatic mode.

AUTO_CMD_OPN could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_OPN and AUTO_CMD_CLS are active at the same time, then the commands CMD_OPN and CMD_CLS get reset.

<table>
<thead>
<tr>
<th>AUTO_CMD_CLS</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Force the valve to close, if the valve is in automatic mode.

AUTO_CMD_CLS could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_OPN and AUTO_CMD_CLS are active at the same time, then the commands CMD_OPN and CMD_CLS get reset.
<table>
<thead>
<tr>
<th>AUTO_CMD_STOP</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Force the valve to stop, if the valve is in automatic mode.
Valve will stop in intermediate position.
AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

<table>
<thead>
<tr>
<th>EN_AUTO</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Enable automatic mode of the PCO_VALV.
PCO_VALV can be set in automatic mode if value of EN_AUTO is TRUE.
PCO_VALV is forced in manual mode if value of EN_AUTO is FALSE.

<table>
<thead>
<tr>
<th>EN_OPN</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Enable opening of the valve.
Valve can be opened if value of EN_OPN is TRUE.
Valve cannot be opened if value of EN_OPN is FALSE.
EN_OPN has no effect in case of emergency open of the valve, e.g. EMCY_OPN.
EN_OPN has no effect on an already opened valve.

<table>
<thead>
<tr>
<th>EN_CLS</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Enable closing of the valve.
Valve can be closed if value of EN_CLS is TRUE.
Valve cannot be closed if value of EN_CLS is FALSE.
EN_CLS has no effect in case of emergency close of the valve, e.g. EMCY_CLS.
EN_CLS has no effect on an already closed valve.

<table>
<thead>
<tr>
<th>EMCY_OPN</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Emergency open of the valve.
If value of EMCY_OPN is TRUE and value of EMCY_CLS is FALSE, the valve will open.
If value of EMCY_OPN is TRUE and the value of EMCY_CLS is TRUE, the valve will close.
If value of EMCY_STOP is TRUE, the valve will stop.
Independent of operation mode of PCO_VALV.
<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALE</td>
<td>-</td>
</tr>
</tbody>
</table>

EMCY_CLS

Emergency close of the valve.

If value of EMCY_CLS is TRUE, the valve will close.
If value of EMCY_OPN is TRUE and value of EMCY_CLS is TRUE, the valve will close.
If value of EMCY_STOP is TRUE, the valve will stop.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALE</td>
<td>-</td>
</tr>
</tbody>
</table>

EMCY_STOP

Emergency stop of the valve.

If value of EMCY_STOP is TRUE, the valve will stop. Highest priority.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALE</td>
<td>-</td>
</tr>
</tbody>
</table>

OPN

Feedback signal from the valve.

If the value of OPN is TRUE, the valve is open.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALE</td>
<td>-</td>
</tr>
</tbody>
</table>

CLS

Feedback signal from the valve.

If the value of CLS is TRUE, the valve is closed.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALE</td>
<td>-</td>
</tr>
</tbody>
</table>

RDY

Valve is ready for operation.

Valve is ready for operation if value of RDY is TRUE.
Value of RDY is FALSE results in an external error.

Independent of operation mode of PCO_VALV.
### REMOTE

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>TRUE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Allow valve to be controlled from SCADA.
Value of REMOTE is TRUE control from SCADA and function block is enabled.
Value of REMOTE is FALSE valve is controlled only from function block.
PCO_VALV will align the block according to feedback signals.
Independent of operation mode of PCO_VALV.

### INHIB_ERR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Suppression of alarms.
If value of INHIB_ERR is TRUE all alarms from PCO_VALV are suppressed.
Independent of operation mode of PCO_VALV.

### EXT_ACK

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

External acknowledge of alarms.
If value of EXT_ACK is TRUE all alarms from PCO_VALV are acknowledged.
Independent of operation mode of PCO_VALV.

### INHIB_FCT_ERR_TIME

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME#5s</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Maximum delay time from command to response from process.
If response is not received within this time limit, a function error will be generated.

### Alarm_1

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 1.
If value of ALARM_1 is TRUE Alarm_1 is active.
Independent of operation mode of PCO_VALV.

### Alarm_2

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 2.
If value of ALARM_2 is TRUE Alarm_2 is active.
Independent of operation mode of PCO_VALV.
<table>
<thead>
<tr>
<th>Alarm_3</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 3.

If value of ALARM_3 is TRUE Alarm_3 is active.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Alarm_4</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Alarm_5</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>Alarm_6</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_VALV.

<table>
<thead>
<tr>
<th>BEHAVE_ON_ERROR</th>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BYTE</td>
<td>0</td>
<td>0 ... 3</td>
<td>-</td>
</tr>
</tbody>
</table>

Actions that need to be executed by the FB at the event of error as set by the user.

The user can set a range of values from 0 ... 3.

BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).

BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.

BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.

BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto/Manual) of PCO_VALV.
OUTPUT_HANDLING

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>1</td>
<td>0 ... 2</td>
<td>-</td>
</tr>
</tbody>
</table>

Behavior of command outputs, CMD_START / CMD_STOP.

The user can set a range of values from 0 ... 2

OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.

OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP) to remain active at RUN or STOP feedback.

With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_VALV.

Output Description

CMD_OPN

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output open to be connected to hardware output.

CMD_CLS

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output close to be connected to hardware output.
### CMD_STOP

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Command output stop, to be connected to hardware output.

### ERR

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE/FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

Common alarm, including:
- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_OPN
- EMCY_CLS
- EMCY_STOP

### ERNO

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Error number

Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in “Standard Function Block Libraries AC500” in “Error Messages of the Function Block Libraries”. 
Input/Output Description

Structure variable for communication to and from SCADA system.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
<th>In-/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCADA.CMDOPN</td>
<td>BOOL</td>
<td>Open command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDCLS</td>
<td>BOOL</td>
<td>Close command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDSTP</td>
<td>BOOL</td>
<td>Stop command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDAUT</td>
<td>BOOL</td>
<td>Auto command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDMAN</td>
<td>BOOL</td>
<td>Manual command</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.CMDRES</td>
<td>BOOL</td>
<td>Acknowledge active alarms</td>
<td>Input</td>
</tr>
<tr>
<td>SCADA.AUTO</td>
<td>BOOL</td>
<td>Valve in automatic mode</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OPEN</td>
<td>BOOL</td>
<td>Valve is open</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.CLOSED</td>
<td>BOOL</td>
<td>Valve is closed</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.REMOTE</td>
<td>BOOL</td>
<td>Valve can be controlled from SCADA</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REMOTE = FALSE → Local operation</td>
<td></td>
</tr>
<tr>
<td>SCADA.READY</td>
<td>BOOL</td>
<td>Valve is ready for operation</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.FNCERR</td>
<td>BOOL</td>
<td>Functional error</td>
<td>Output</td>
</tr>
<tr>
<td>Parameter</td>
<td>Data type</td>
<td>Description</td>
<td>In-/Output</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>SCADA.EXTERR</td>
<td>BOOL</td>
<td>External error (is generated when the valve is not ready)</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL1</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 1</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL2</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 2</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL3</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 3</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL4</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 4</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL5</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 5</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.AL6</td>
<td>BOOL</td>
<td>Auxiliary alarm no. 6</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg1 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_VALV</td>
<td>Output</td>
</tr>
<tr>
<td>SCADA.OSMsg2 *)</td>
<td>WORD</td>
<td>Word representing the status of the PCO_VALV</td>
<td>Output</td>
</tr>
</tbody>
</table>

*) structure described separately

### SCADA.OSMsg1

<table>
<thead>
<tr>
<th>Data type</th>
<th>Default value</th>
<th>Range</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Word representing the status of the PCO_VALV.

### Bit

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Valve is open</td>
</tr>
<tr>
<td>1</td>
<td>Valve is closed</td>
</tr>
<tr>
<td>2</td>
<td>Valve opening</td>
</tr>
<tr>
<td>3</td>
<td>Valve closing</td>
</tr>
<tr>
<td>4</td>
<td>External error (Not ready)</td>
</tr>
<tr>
<td>5</td>
<td>Functional error</td>
</tr>
<tr>
<td>6</td>
<td>Valve released for opening</td>
</tr>
<tr>
<td>7</td>
<td>Valve released for closing</td>
</tr>
<tr>
<td>8</td>
<td>Local operation (Not remote)</td>
</tr>
<tr>
<td>9</td>
<td>Common alarm (External alarm + Functional alarm + EMCY_OPN/EMCY_CLS/EMCY_STOP + ALARM_1 ... ALARM_6)</td>
</tr>
<tr>
<td>10</td>
<td>Unacknowledged alarm</td>
</tr>
<tr>
<td>11</td>
<td>Valve in automatic mode</td>
</tr>
<tr>
<td>12</td>
<td>Not used</td>
</tr>
<tr>
<td>13</td>
<td>Valve not released for automatic mode</td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
</tr>
<tr>
<td>15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
Word representing the status of the PCO_VALV.
This includes the status of emergency status and alarm status.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Emergency open</td>
</tr>
<tr>
<td>1</td>
<td>Emergency close</td>
</tr>
<tr>
<td>2</td>
<td>Auxiliary alarm 1</td>
</tr>
<tr>
<td>3</td>
<td>Auxiliary alarm 2</td>
</tr>
<tr>
<td>4</td>
<td>Auxiliary alarm 3</td>
</tr>
<tr>
<td>5</td>
<td>Auxiliary alarm 4</td>
</tr>
<tr>
<td>6</td>
<td>Auxiliary alarm 5</td>
</tr>
<tr>
<td>7</td>
<td>Auxiliary alarm 6</td>
</tr>
<tr>
<td>8</td>
<td>Emergency stop</td>
</tr>
<tr>
<td>9</td>
<td>Not used</td>
</tr>
<tr>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>11</td>
<td>Not used</td>
</tr>
<tr>
<td>12</td>
<td>Not used</td>
</tr>
<tr>
<td>13</td>
<td>Not used</td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
</tr>
<tr>
<td>15</td>
<td>Not used</td>
</tr>
</tbody>
</table>
2 Index