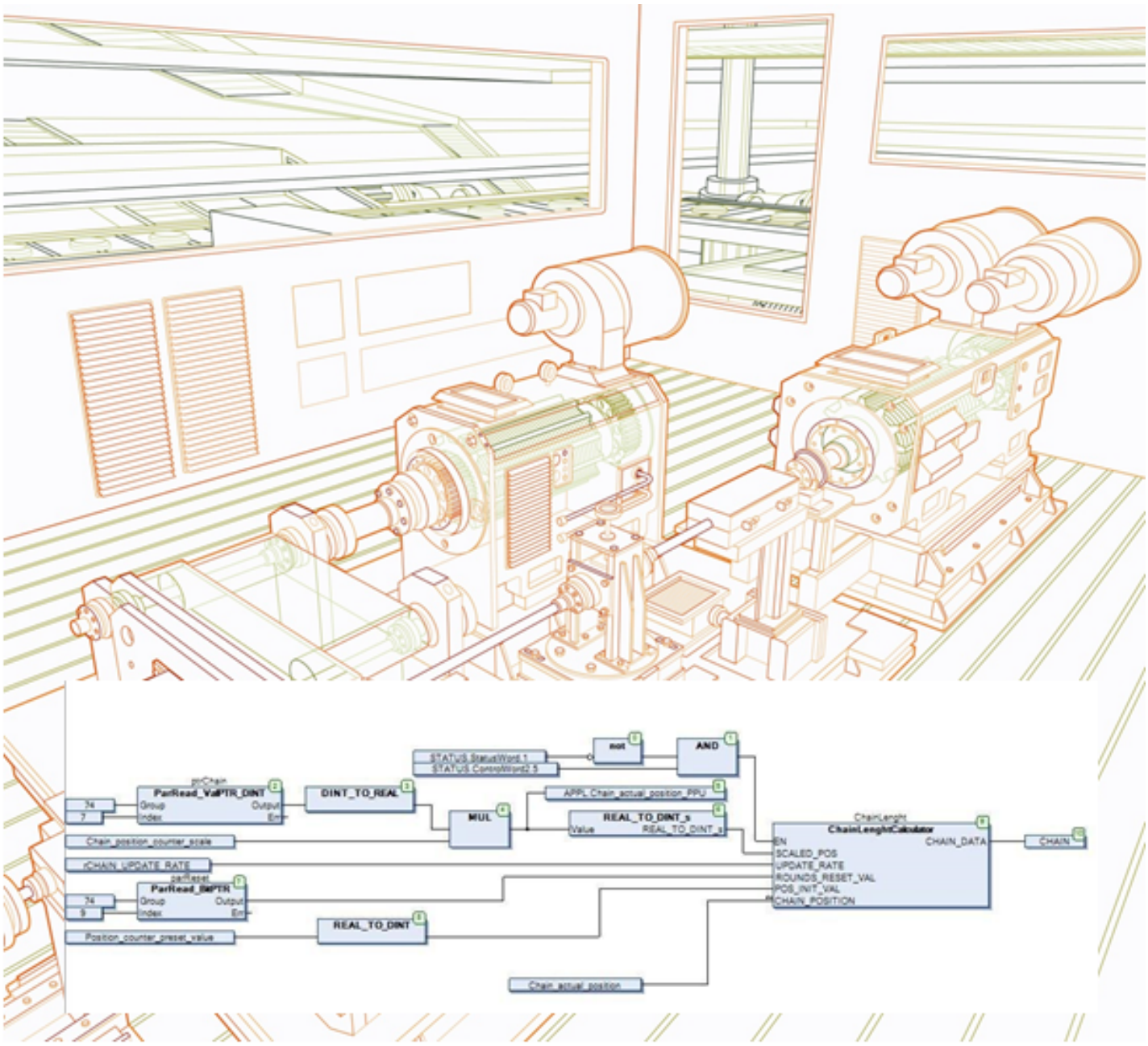
ABB INDUSTRIAL DRIVES

Drive application programming (IEC 61131-3)

Programming manual



Drive application programming (IEC 61131-3)

Programming manual

Table of contents

3AUA0000127808 Rev G
EN
Original instructions
EFFECTIVE: 2021-12-31

Table of contents

1 Introduction

Contents of this chapter 15
Compatibility 15
Target audience 15
Safety instructions 16
Purpose of the manual 16
Terms and abbreviations 16
Related documents 17
Cybersecurity disclaimer 17
Notes 18

2 Getting started

Contents of this chapter 19
Settings up the programming environment 19

3 Overview of drive programming

Contents of this chapter 21
Drive application programming 21
System diagram 22
Programming work cycle 23
Special tasks 23
Programming languages and modules 23
Libraries 24
Program execution 24
DriveInterface 24
ApplicationParametersandEvents 24

4 Creating application program

Contents of this chapter 25
Creating a new project 25
Updating project information 27
Appending a new POU 29
Writing a program code 31
Continuous function chart (CFC) program 31
Adding elements 31
Setting the execution order of the elements 34
Adding comments to a CFC program 34
Declaring variables 35
Adding inputs and outputs 36
CFC program 36
Preparing a project for download 37
Establishing online connection to the drive 37
Downloading the program to the drive 40
Creating a boot project 41
Opening a project archive 41

Executing the program 45

5 Features

Contents of this chapter 47
Device handling 47
Viewing device information 48
Viewing device serial number 49
Upgrading or adding a new device 51
Changing an existing device 51
Viewing software updates 51
Program organization units (POU) 51
Data types 52
Drive application programming license 53
Application download options 53
Removing the application from the target 54
Retain variables 55
Task configuration 56
Adding tasks 56
Monitoring tasks 58
Uploading and downloading source code 59
Adding symbol configuration 59
Debugging and online changes 60
Safe debugging 60
Reset options 61
Memory limits 61
CPU limitation 62
Application loading package 63
Loading firmware to a drive 64

6 DriveInterface

Contents of this chapter 71
Implementing DriveInterface 71
Selecting the parameter set 72
Viewing parameter mapping report 73
Mapping example 73
Updating drive parameters from installed device 74
Updating drive parameters from parameters file 75
Setting parameter view 75

7 Application parameters and events

Contents of this chapter 77
Application parameters and events 77
Parameter manager 77
Creating parameter groups 78
Importing and exporting parameters 78
Creating parameters 79
Parameter settings 80
Scaling 82
Tool/Fieldbus 32-bit interface 82
Fieldbus 16-bit interface 82
Testing for scaling 82

Linking parameter to application code 82
Parameter types 83
Parameter families 84
Selection lists 85
Units 86
Application events 87

8 Configuring extension I/O modules

Contents of this chapter 89
Configuring extension I/O module 89
FEA-03 89
Node numbers 91
Selecting input signal type 92
FDCO 93
Extension I/O in drive application program 94
Adding F-series module 94
Setting module data 96
Adding node number 96
I/O mapping variables 96
Using F-series I/O from the application 96
Adding bus fault control 98
FIO-01 Module data 99
FIO-01 Channel descriptions 99
FIO-11 Module data 101
FIO-11 Channel descriptions 101
FAIO-01 Module data 104
FAIO-01 Channel descriptions 104
Fault codes 106

9 Libraries

Contents of this chapter 107
Library types 107
Adding a library to the project 108
Creating a new library 110
Installing a new library 112
Managing library versions 113
Configuring a library with WIBU license 114

10 Practical examples and tips

Contents of this chapter 115
Solving communication problems 115
Solving other problems 116

11 Unsupported features

Contents of this chapter 117
Unsupported features 117

12 ABB drives system library

Contents of this chapter 119
Overview 119

Function blocks of the system library 120
Event function blocks 122
EVENT 122
Summary 122
Connections 122
Description 122
ReadEventLog 124
Summary 124
Connections 124
Description 125
Parameter change function blocks 126
PAR_UNIT_SEL 126
Summary 126
Connections 126
Description 126
PAR_SCALE_CHG 127
Summary 127
Connections 127
Description 127
External 32-bit scaling 128
External 16-bit scaling 128
Parameter limit change 129
PAR_LIM_CHG_DINT 129
Summary 129
Connections 129
Description 129
PAR_LIM_CHG_REAL 130
Summary 130
Connections 130
Description 130
PAR_LIM_CHG_UDINT 131
Summary 131
Connections 131
Description 131
Parameter default value change 132
PAR_DEF_CHG_DINT 132
Summary 132
Connections 132
Description 132
PAR_DEF_CHG_REAL 133
Summary 133
Connections 133
Description 133
PAR_DEF_CHG_UDINT 134
Summary 134
Connections 134
Description 134
Parameter decimal display 135
PAR_DISP_DEC 135
Summary 135
Connections 135
Description 135

PAR_REFRESH 136
Summary 136
Connections 136
Description 136
Parameter protection 137
PAR_PROT 137
Summary 137
Connections 137
Description 137
PAR_GRP_PROT 138
Summary 138
Connections 138
Description 138
Parameter read function blocks 139
ParReadBit 139
Summary 139
Connections 139
Description 139
ParRead_INT 140
Summary 140
Connections 140
Description 140
ParRead_DINT 141
Summary 141
Connections 141
Description 141
ParRead_REAL 142
Summary 142
Connections 142
Description 142
ParRead_UDINT 143
Summary 143
Connections 143
Description 143
Parameter write function blocks 144
ParWriteBit 144
Summary 144
Connections 144
Description 144
ParWrite_DINT 145
Summary 145
Connections 145
Description 145
ParWrite_INT 146
Summary 146
Connections 146
Description 146
ParWrite_REAL 147
Summary 147
Connections 147
Description 147

ParWrite_UDINT 148
Summary 148
Connections 148
Description 148
Pointer parameter read function block 149
ParRead_BitPTR 149
Summary 149
Connections 149
Description 149
ParRead_ValPTR_DINT 150
Summary 150
Connections 150
Description 150
ParRead_ValPTR_REAL 151
Summary 151
Connections 151
Description 151
ParRead_ValPTR_UDINT 152
Summary 152
Connections 152
Description 152
GetPtrParConf 153
Summary 153
Connections 153
Description 153
Set pointer parameter to IEC variable function blocks 155
ParSet_BitPTR_IEC 155
Summary 155
Connections 155
Description 155
ParSet_ValPTR_IEC_DINT 155
Summary 156
Connections 156
Description 156
ParSet_ValPTR_IEC_REAL 157
Summary 157
Connections 157
Description 157
ParSet_ValPTR_IEC_UDINT 158
Summary 158
Connections 158
Description 158
Set pointer parameter to parameter function blocks 159
ParSet_BitPTR_Par 159
Summary 159
Connections 159
Description 159
ParSet_ValPTR_Par 160
Summary 160
Connections 160
Description 160

System time function blocks 161
SYS_TIME 161
Summary 161
Connections 161
Description 161
SYS_TIME_UDINT 163
Summary 163
Connections 163
Description 163
Task time level function block 165
UsedTimeLevel 165
Summary 165
Connections 165
Description 165
Error codes 166

13 ABB D2D function blocks

Contents of this chapter 167
Introduction to ABB D2D function blocks 167
D2D communication library 168
D2D block error codes 168
Data read/write blocks 169
DS_ReadLocal 169
Summary 169
Connections 169
Description 169
DS_WriteLocal 170
Summary 170
Connections 170
Description 170
D2D communication blocks 171
General 171
D2D_TRA 171
Summary 171
Connections 171
Description 172
D2D_REC 173
Summary 173
Connections 173
Description 173
D2D_TRA_REC 175
Summary 175
Connections 175
Description 175
D2D_TRA_MC 177
Summary 177
Connections 177
Description 177
D2D configuration blocks 179
D2D_Conf 179
Summary 179
Connections 179

Description 179
Master use 179
Follower use 180
D2D_Conf_Token 181
Summary 181
Connections 181
Description 181
D2D_Master_State 183
Summary 183
Connections 183
Description 183
Examples: D2D blocks 184
Example 1: D2D_TRA / D2D_REC blocks 184
Example 2: Token send configuration blocks 184

14 ABB drives standard library

Contents of this chapter 187
Overview 187
Basic functions 188
BGET 188
Summary 188
Connections 188
Function 188
BSET 189
Summary 189
Connections 189
Function 189
DEMUX 190
Summary 190
Connections 190
Function 190
DEMUXM 191
Summary 191
Connections 191
Function 191
MUX 192
Summary 192
Connections 192
Function 192
MUXM 193
Summary 193
Connections 193
Function 193
PACK 194
Summary 194
Connections 194
Function 194
SR_D 195
Summary 195
Connections 195
Function 195
Truth table 195

SWITCH 196
Summary 196
Connections 196
Function 196
SWITCHC 197
Summary 197
Connections 197
Function 197
UNPACK 198
Summary 198
Connections 198
Function 198
Special functions 199
Drive control 199
Summary 199
Connections 199
Function 201
Limiting 201
Filter 201
Summary 201
Connections 201
Function 202
Function generator 202
Summary 202
Connections 203
Function 203
Interpolation 203
Balancing 204
Limiting 204
Integrator 204
Summary 204
Connections 205
Function 205
Clearing 206
Tracking 206
Limiting 206
Lead lag 206
Summary 206
Connections 206
Function 207
Balancing 207
Reset 207
Motor potentiometer 208
Summary 208
Connections 208
Function 208
Clearing 209
Tracking 209
Limiting 209
PID 209
Summary 209
Connections 209

Function 210
Smooth transfer 211
Filtering 212
Tracking 212
Limitation function 212
Limiting 213
Ramp 213
Summary 213
Connections 213
Function 214
Tracking 214
Limiting 215

Further information

1

Introduction

Contents of this chapter

This chapter describes the contents of the manual. It also contains information on the compatibility, safety and intended audience.

Compatibility

This manual applies to ABB drives equipped with the application programming functionality. The drives can be ordered with the application programming functionality. The drive must be equipped with N8010 Application programming license on ZMU-02.

The following drives are compatible with application programming function.

- ACS880
- DCS880
- DCT880
- HES880

This manual is compatible with the following product releases:

- Drive Application Builder 1.2
- Drive composer pro 2.7 or later

For more details of compatibility information, refer the corresponding drive software release notes or contact your ABB representative.

Target audience

This manual is intended for a personnel performing drive application programming or for understanding the programming environment capabilities. The reader of the manual is

expected to have basic knowledge of the drive technology and programmable devices (drive and PC) and programming methods.

Safety instructions

Follow all safety instructions delivered with the drive.

- Read the complete safety instructions before you load and execute the application program on the drive or modify the drive parameters. The complete safety instructions are delivered with the drive as either part of the hardware manual, or, in the case of ACS880 multidrives, as a separate document.
- Read the firmware function-specific warnings and notes before changing parameter values. These warnings and notes are included in the parameter descriptions presented in chapter Parameters of the firmware manual.

WARNING!
Ignoring the following instruction can cause physical injury or damage to the equipment.

Do not make changes to drive in the online mode or download programs while the drive is running to avoid damages to the drive.

Purpose of the manual

This manual gives basic instructions on the drive-based application programming using Drive Application Builder programming tool. The programming tool is the international IEC 61131-3 programming standard. The online help of Drive Application Builder contains more detailed information of the IEC languages, programming methods, editors and tool commands.

Terms and abbreviations

Term	Description
BCU	Type of control unit
Drive	Frequency converter for controlling AC motors
EMC	Electromagnetic compatibility
EMI	Electromagnetic interference
FEN-01	Optional TTL incremental encoder interface module
FEN-11	Optional TTL absolute encoder interface module
FEN-21	Optional resolver interface module
FEN-31	Optional HTL incremental encoder interface module
FIO-11	Optional analog I/O extension module
FPTC-01	Optional thermistor protection module
FPTC-02	Optional ATEX-certified thermistor protection module for potentially explosive atmospheres
Frame, frame size	Physical size of the drive or power module
FSO-12, FSO-21	Optional functional safety modules
IGBT	Insulated gate bipolar transistor
Inverter unit	Inverter module(s) under control of one control unit, and related components. One inverter unit typically controls one motor.
Power module	Common term for drive module, inverter module, supply module, brake chopper module etc.

Term	Description
RFI	Radio-frequency interference
STO	Safe torque off (IEC/EN 61800-5-2)
Supply unit	Supply module(s) under control of one control unit, and related components.

Related documents

Name	Code
Drive manuals and guides	
Drive application programming manual (IEC 61131-3)	3AUA0000127808
ACS880 primary control program firmware manual	3AUA0000085967
Option manuals and guides	
FDCO-01/02 DDCS communication modules user's manual	3AUA0000114058
FEA-03 F-series extension adapter user's manual	3AUA0000115811
FAIO-01 analog I/O extension module user's manual	3AUA0000124968
Digital I/O Extension FIO-01 user's manual	3AFE68784921
Analog I/O Extension FIO-11 user's manual	3AFE68784930
Tool and maintenance manuals	
Drive composer PC tool user's manual	3AUA0000094606

Cybersecurity disclaimer

This product is designed to be connected to and to communicate information and data via a network interface. It is your sole responsibility to provide and continuously ensure a secure connection between the product and your network or any other network (as the case may be). You shall establish and maintain any appropriate measures (such as but not limited to the installation of firewalls, application of authentication measures, encryption of data, installation of anti-virus programs, etc.) to protect the product, the network, its system and the interface against any kind of security breaches, unauthorized access, interference, intrusion, leakage and/or theft of data or information. ABB Ltd and its affiliates are not liable for damages and/or losses related to such security breaches, any unauthorized access, interference, intrusion, leakage and/or theft of data or information.

Although ABB provides functionality testing on the products and updates that we release, you should institute your own testing program for any product updates or other major system updates (to include but not limited to code changes, configuration file changes, third party software updates or patches, hardware exchanges, etc.) to ensure that the security measures that you have implemented have not been compromised and system functionality in your environment is as expected. This also applies to the operating system. Security measures (such as but not limited to the installation of latest patches, installation of firewalls, application of authentication measures, installation of anti-virus programs, etc.) are in your responsibility. You have to be aware that operating systems provide a considerable number of open ports that should be monitored carefully for any threats.

Notes

- To support the main functionality of Drive Application Builder, open the specific port, services and software in your computer. The below table shows the required list of ports, services and software.

443	https	WIBU license protection, User personal data storage	ABBDrivesLicenseManager
443	https	Installation file download	DriveApplicationBuilderSetup
80	http	Transfer application files to target and to debug device	DriveDA
443	https	SVN integration	DriveApplicationBuilder
22350	TCP/UDP	Codemeter service	Codemeter
22352	TCP/UDP	Codemeter service	Codemeter

- ABB recommends to use secured protocols on connecting to external interfaces. For example, use secured HTTP (HTTPS) connection while using SVN functionality.

2

Getting started

Contents of this chapter

This chapter provides information on how to setup a program and how to upgrade, change and view device information in Drive Application Builder.

Settings up the programming environment

The following software installations are required for programming the drives.

- ABB drives with Drive application programming license (N8010)
- Drive Application Builder 1.2
- ACS-AP-x control panel and micro USB cable
- Drive composer pro 2.7 or later

For details of the version, refer the corresponding drive software release notes or contact your ABB representative.

Note: Install Drive composer pro and then proceed with the installation of Drive Application Builder. In case of any issues, contact your ABB representative.

The Drive composer pro enables setting and monitoring of the drive parameters and signals. The control panel acts as a USB/RS485 converter between Drive Application Builder, Drive composer pro and the drive.

To setup drive programming environment follow the below pre-requisites and installation steps.

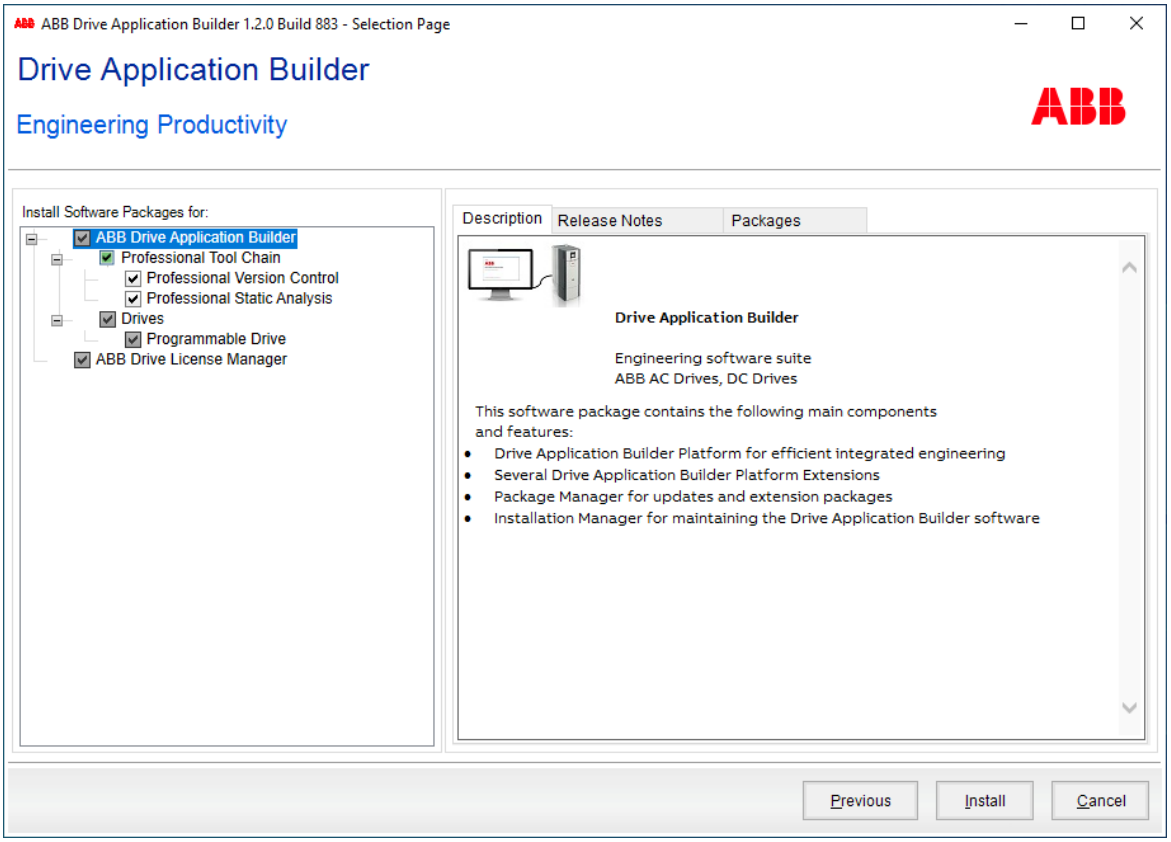
Pre-requisites:

- The Drive Application Builder supports Windows 10 (32-bit and 64-bit versions) operating systems.
- You must have administrator user rights to install Drive Application Builder.

Installation steps:

1. Install Drive composer pro to enable communication with the target drive. For more details, see *Drive composer user's manual* (3AUA0000094606 [English]).
2. In Drive composer pro, go to **System info** → **Products/Licenses**, check that the drive has an active IEC programming license and the drive firmware version is correct.
   For details of version, refer the corresponding drive software release notes or contact your ABB representative.

Install Drive Application Builder according to the instruction guide included in the installation media of Drive Application Builder. All drive application programming related components are automatically installed.



To allow parallel communication with Drive Application Builder and Drive composer pro, follow these steps:

1. In the main menu of Drive composer pro, click **View** and select **Settings**.
2. In the Settings window, enable Share connection check-box and click **Save** to connect with Drive Application Builder.

After configuring the settings, restart Drive composer pro. Drive composer now connects to the drive and allows sharing the connection with Drive Application Builder.

3

Overview of drive programming

Contents of this chapter

This chapter provides an overview of drive programming environment and a typical work cycle of drive application programming.

Drive application programming

The drive application programming function allows you to add your own program code to the drive using the Drive Application Builder programming tool. The programming method and languages are based on the IEC 61131-3 programming standard.

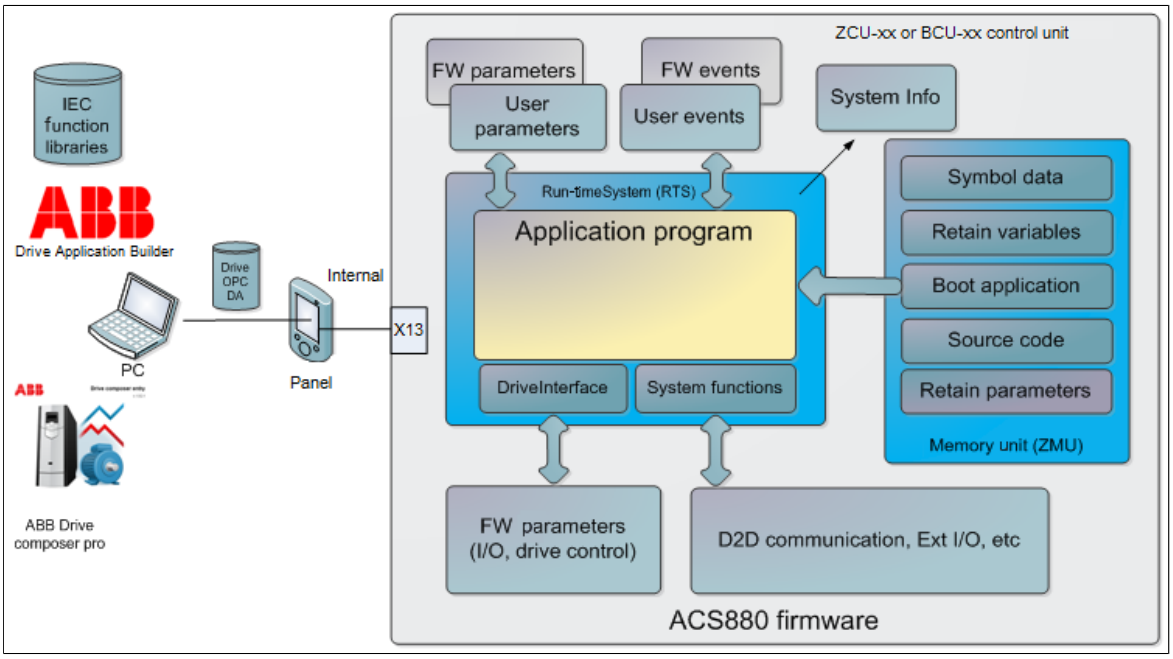
With the drive application programming, you can create application specific features on top of the drive firmware functionality. You can utilize the standard and extension I/O and communication interfaces of the drive along with the appropriate firmware signals. The program is executed in parallel with the drive control tasks using the same hardware resources.

In addition, you can create your own parameters and events (faults and warnings) that are visible on the ACS-AP-x control panel and in the Drive composer pro/entry commissioning tools.

Note: For using Drive Application Builder online with the drive, enable the drive application programming license in the target drive. See chapter *Creating application program (page 25)*.

System diagram

The following simplified system diagram shows the application programming environment in the same control unit as the drive firmware.



The following list describes the main components of application programming.

Drive control unit

- Run-time system (RTS) executes the application program.
- DriveInterface allows input/output mapping between the application program and drive firmware parameters.
- System function library enables access to the drive system services (parameters/events/drive-to-drive communication, extension I/O).
- User made parameters.
- User made events (fault, warnings).
- Drive System info includes version information of the application program.
- Drive firmware parameters with I/O controls.
- D2D function blocks enable drive to drive communication, I/O extension modules, and so on for application programming.

Drive memory unit

- Creates a permanent version of the application program (Boot application).
- Retains values of the application program variables.
- Consists of application source code (Note that the size of the memory is limited).
- Includes symbol and address information of the application program variables for monitoring purposes.

PC tool programs

- Drive Application Builder for application program development and online operations.

- Drive composer pro for drive parameter, signal, event log monitoring and settings.
- Application program function libraries (for example, ABB standard library).
- The USB/ACS-AP-x control panel enables communication between the Drive Application Builder, Drive composer pro and the drive.

Programming work cycle

The following steps describe a typical work cycle of the drive application programming task.

1. In the Devices tree, do the following:
   - create a new project
   - add objects
   - define the target and first program module.
2. In the DriveInterface object, define the interface to drive firmware parameters (I/O access, drive control).
3. In Devices tree, define user parameters and events (ApplicationParametersandEvents) module.
4. Develop the program structure and coding program units.
5. Define the program execution task configuration editor.
6. Compile and load the code using **Build** menu.
7. In the Online menu, do the following:
   - Select Create boot application if new parameters, mappings, events or task configuration are added to the program.
   - Debug the program code (stepping, forcing variables and breakpoints).
8. In the View menu, select Watch window to monitor program variables in Drive Application Builder and Drive composer pro.
9. Repeat steps 2 to 8 to test the program.

Special tasks

The following special tasks are part of the drive application programming tasks.

1. Using Online menu, save or restore the source code to the permanent memory of a drive.
2. Save the drive IEC symbol data to the permanent memory of a drive from the Devices tree using Add Symbol configuration object to the tree option.
3. In the Application properties window or Project information, create a name and version of the application.
4. In the Online menu, select **Reset origin** to remove the application from the target.

Programming languages and modules

The programming environment supports programming languages as specified in the IEC 61131-3 standard with some useful extensions.

The following programming languages are supported:

- Ladder diagram (LD)
- Function block diagram (FBD)
- Structured text (ST)
- Instruction list (IL)
- Sequential function chart (SFC)
- Continuous function chart (CFC), normal and page-oriented CFC editor

A program can be composed of multiple modules like functions, function blocks and programs. Each module can be implemented independently with the above mentioned languages. Each language has its own dedicated editors.

For more information on programming languages, see chapter *Features (page 47)*.

Libraries

The program modules can be implemented in the projects or imported into libraries. A library manager is used to install and access the libraries.

The two main types of libraries are:

- Local libraries (IEC language source code, for example, AS1LB_Standard_ACS880_V3_5)
- External libraries (external implementation and source code, for example, AY1LB_System_ACS880_V3_5)

Local libraries include source code or can be compiled. If the library is compiled, source code is not included in the library.

For more information, see chapter *Libraries (page 107)*.

Program execution

The program is executed on the same central processing unit (CPU) as the other drive control tasks. In real time applications, programs are typically executed periodically as cyclic tasks. The programmer can define the cyclic task interval. For more information, see chapter *Features (page 47)*.

DriveInterface

The DriveInterface object enables input and output mapping between application program and drive firmware using drive firmware parameters used in the application program. The list of parameters may be different for each drive firmware versions. For more details on implementing the DriveInterface and updating parameter list, see chapter *DriveInterface (page 71)*.

ApplicationParametersandEvents

The ApplicationParameterandEvents Manager (APEM) object allows creating application parameter groups, parameters, parameter types, parameter families, units and application events for the drive in Drive Application Builder environment. For more details on how to create parameter related tasks and application events, see chapter *Application parameters and events (page 77)*.

4

Creating application program

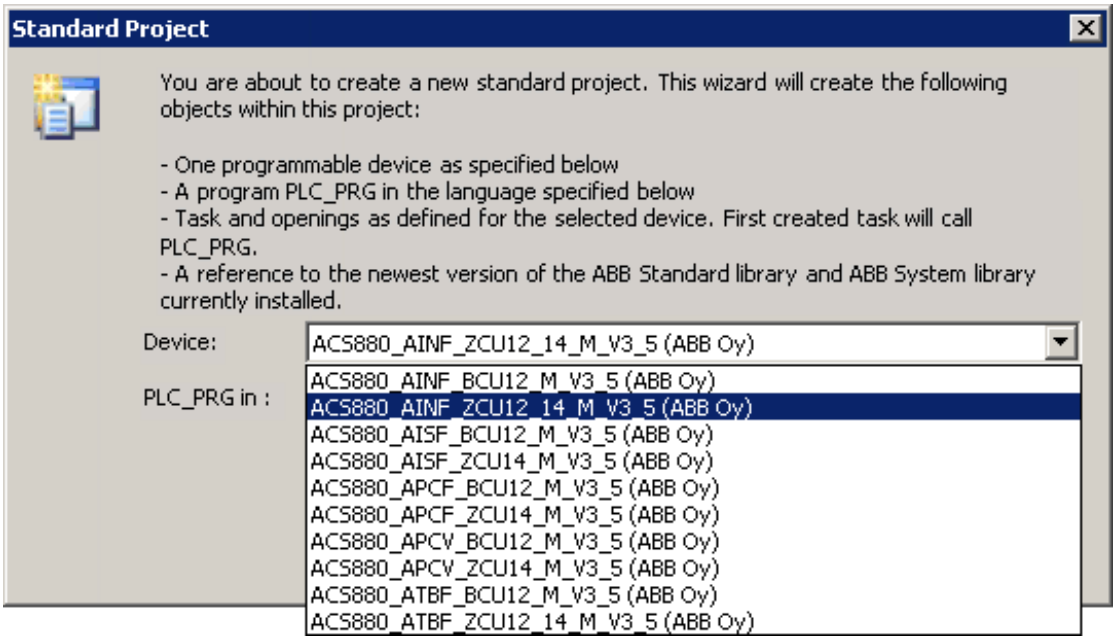
Contents of this chapter

This chapter describes the procedure to create application program.

Creating a new project

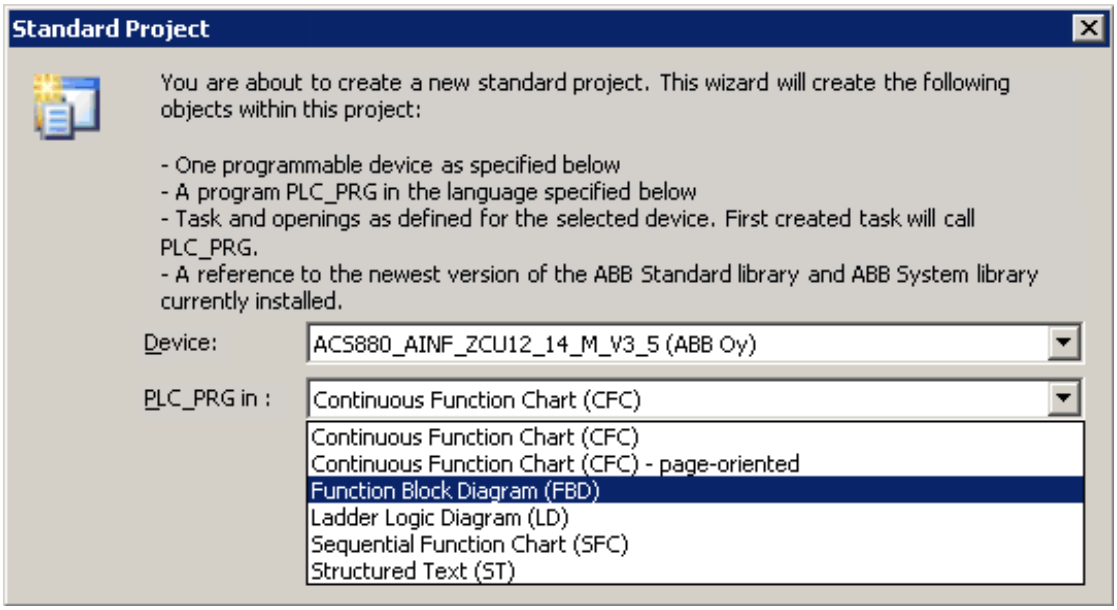
After starting Drive Application Builder programming environment, you can create a new project.

1. In the Start Page, click **New Project** or in the main menu, go to **File** and select **New Project**.
2. In the New Project dialog, select the required project template and click **OK**.
   You can rename the project in Name field and select the desired Location in the file system.
3. In the Standard Project dialog, select the type of control unit in Device drop-down list.

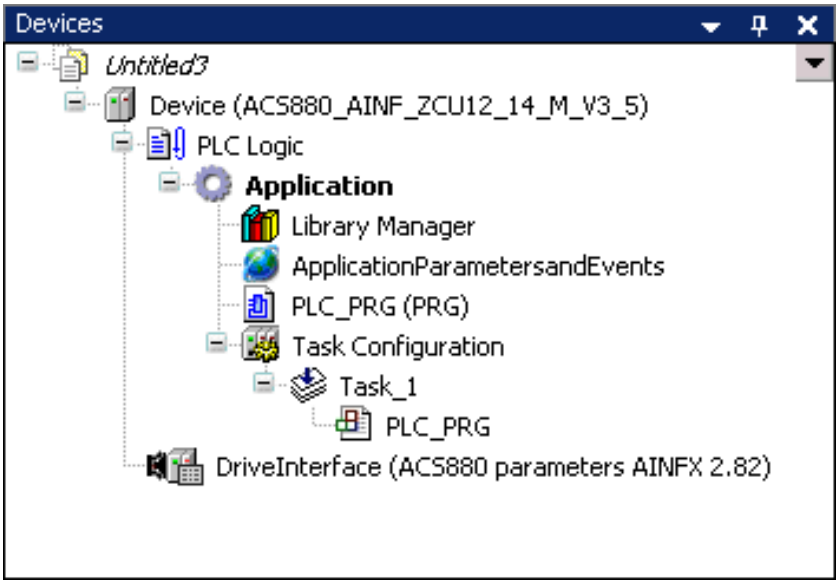


Check the control unit type of the target drive either from the unit itself from the hardware manual of drives or contact your local ABB representative.

4. In the PLC_PRG in drop-down list, select a programming language and click **OK**. You can later add program modules made with other languages to the project.



A simple project for ACS880 target drive is created in the Devices tree.



The Devices tree includes:

- PLC Logic
- DriveInterface for firmware signal and parameter mapping
- Application (for example, you can add the following objects under Application)
  - Library Manager for installing function libraries
  - ApplicationParametersandEvents for creating user parameters and events
  - Program organization units (POUs)
  - Task Configuration module for defining in which task the POUs are executed
  - Text list
  - Symbol configuration
  - Global variable list
  - Data type units (DUT)

Updating project information

You can update company name and version number for the application program in the Project Information window. This information is visible in Drive composer tool and ACS-AP-x control panel in the System info display. It helps to identify the loaded application without the Drive Application Builder. You can also name the application from the application tool.

To update project information in Drive Application Builder, follow these steps:

1. In the main menu, go to **Project** and select **Project Information**.
2. In the Project Information dialog, go to Summary tab and update the desired information and click **OK**.
   The updated project information is not loaded to the target application. Further steps explain how to copy this information to the application information fields.
3. In the Devices tree, right click **Applications** and select **Properties**.

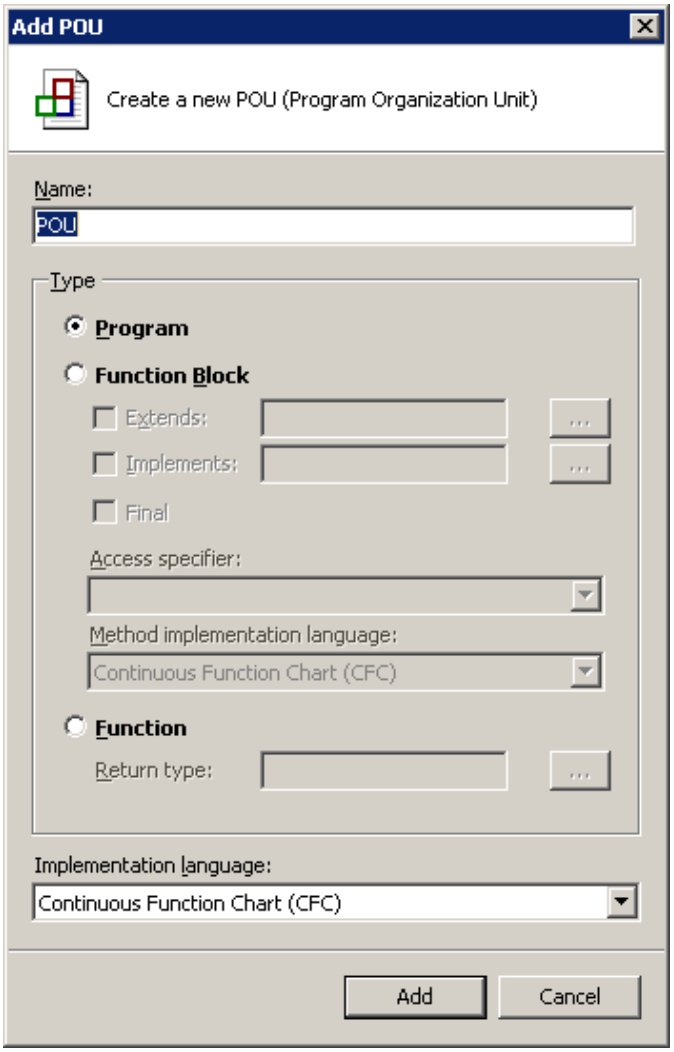
Properties dialog is displayed.

4. Click **Information** tab and click **Reset to values from project information** and then click **OK**.
   The Drive Application Builder version and project identification code are registered automatically.

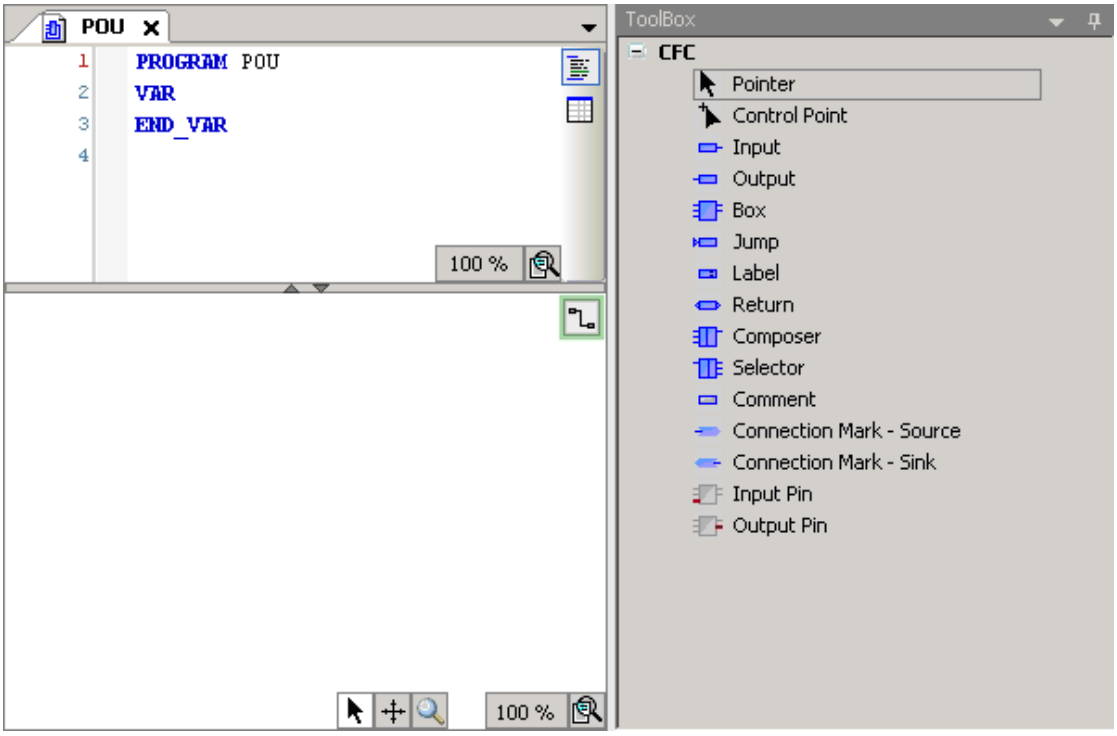
Appending a new POU

To append a new POU, follow these steps:

1. In the Devices tree, right-click Application and select **Add object**.
2. Select POU and click **Add object**.
3. In the Add POU dialog, Name the POU, select the Type of the POU and the used implementation language and then click **Add**.



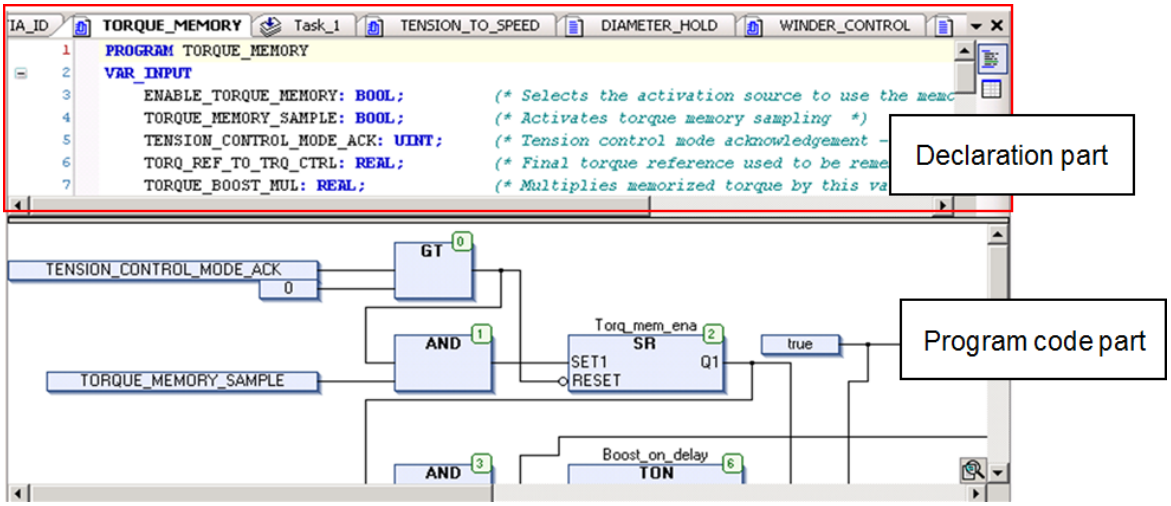
The appended POU, xxx (PRG) is added to the Devices tree under application. The POU dialog is displayed with the declaration part and the program code.


POU
PROGRAM POU
VAR
END_VAR
100 %
100 %
ToolBox
CFC
Pointer
Control Point
Input
Output
Box
Jump
Label
Return
Composer
Selector
Comment
Connection Mark - Source
Connection Mark - Sink
Input Pin
Output Pin

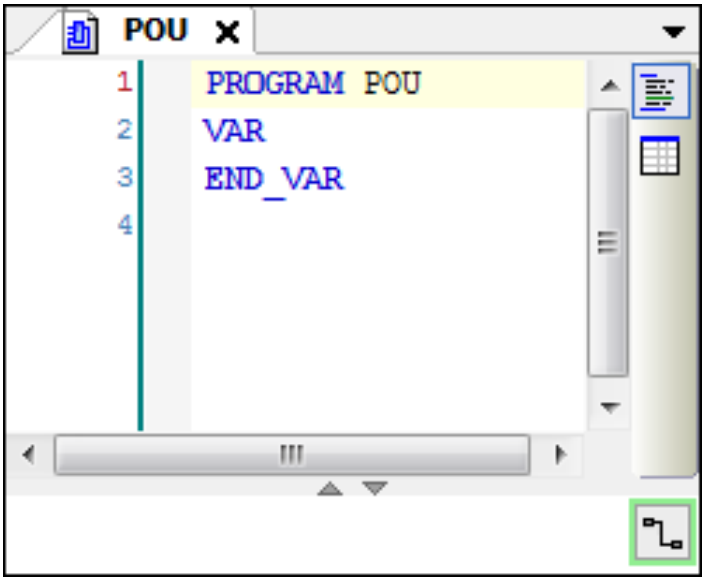
Writing a program code

A program organization unit (POU) is a unit, object or area where you can write the program code. The units can be created either directly under the Applications in the Devices tree or in a separate POUs window (**View** -> **POUs** or click **POUs** in the lower left corner).

The POU includes a declaration part (the upper window) and a program code part (the lower window).



There are two different types of views for declaration part: a textual view and tabular view. You can switch between these views by clicking the buttons.



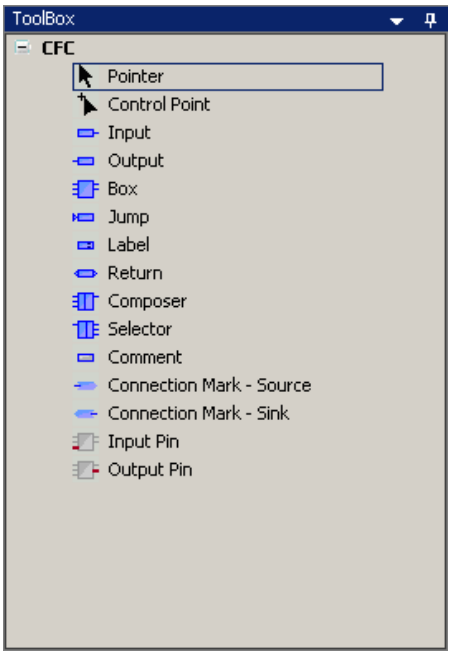
Continuous function chart (CFC) program

The following sections show how to create a new project in the CFC implementation language.

Adding elements

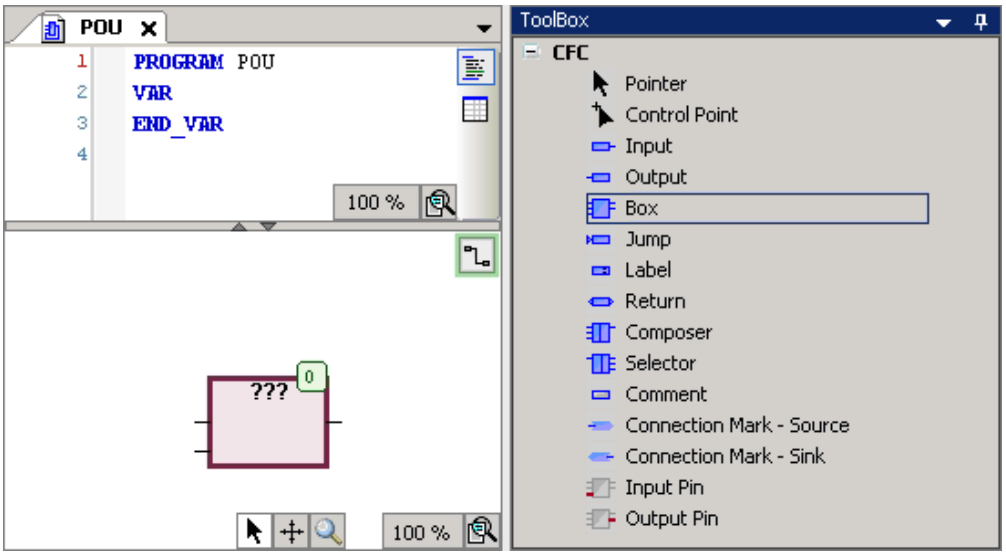
1. In the Devices tree, expand Application and select **xxx(PRG)**.
2. In the main menu, click **View** and select **ToolBox**.

ToolBox components are displayed and are used to add a CFC scheme.

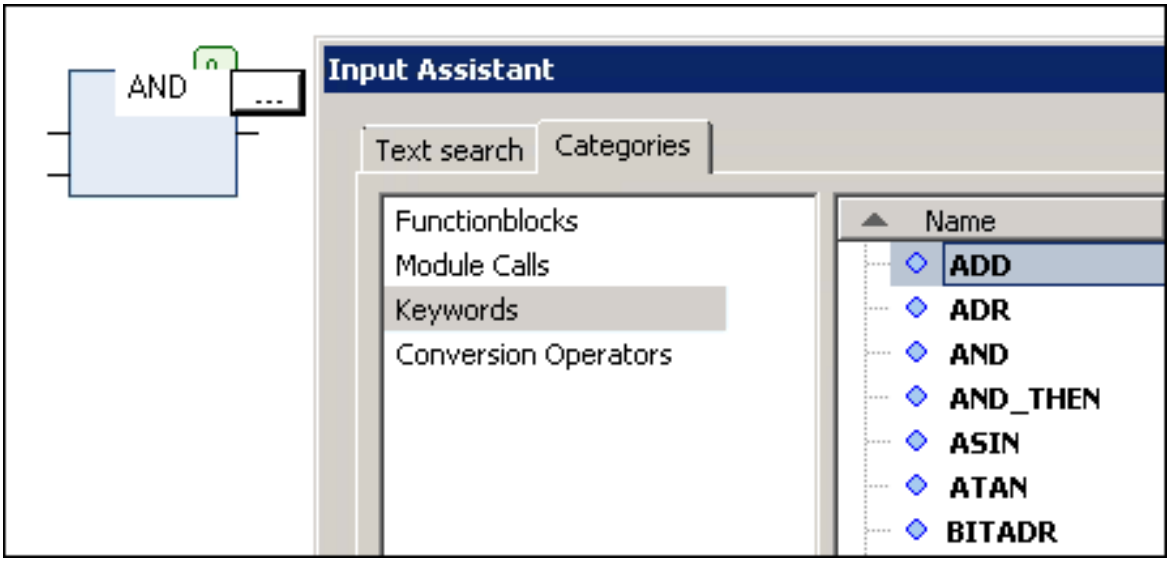


If an empty ToolBox list is already displayed on the right side of the window, double-click the xxx (PRG) to display the Toolbox and the POU window.

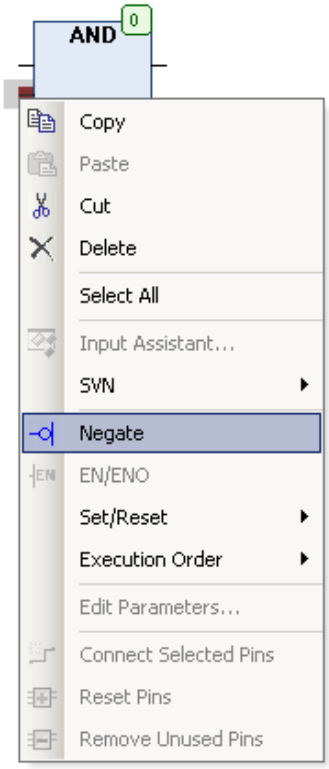
3. To add SEL and AND elements (logic operators, functions), use the Box element in the ToolBox list.
   In the ToolBox list, drag and drop the Box into the program code area.



4. Enter the name of the function or operand in the ??? field.
   - You can also use Input Assistant to find the function, keyword, and operator. To start Input Assistant, click the button or press F2.
   - The number in the upper right corner of the white box indicates the execution order of the function.



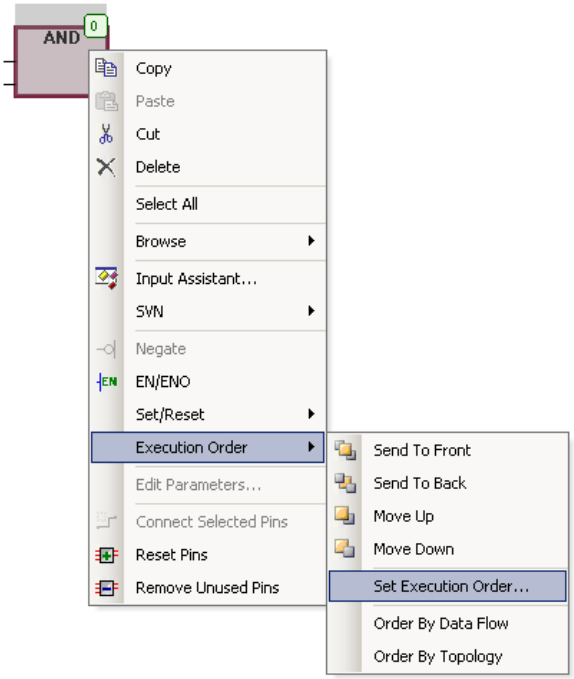
5. Right-click on input or output element and select **Negate** to invert.



Setting the execution order of the elements

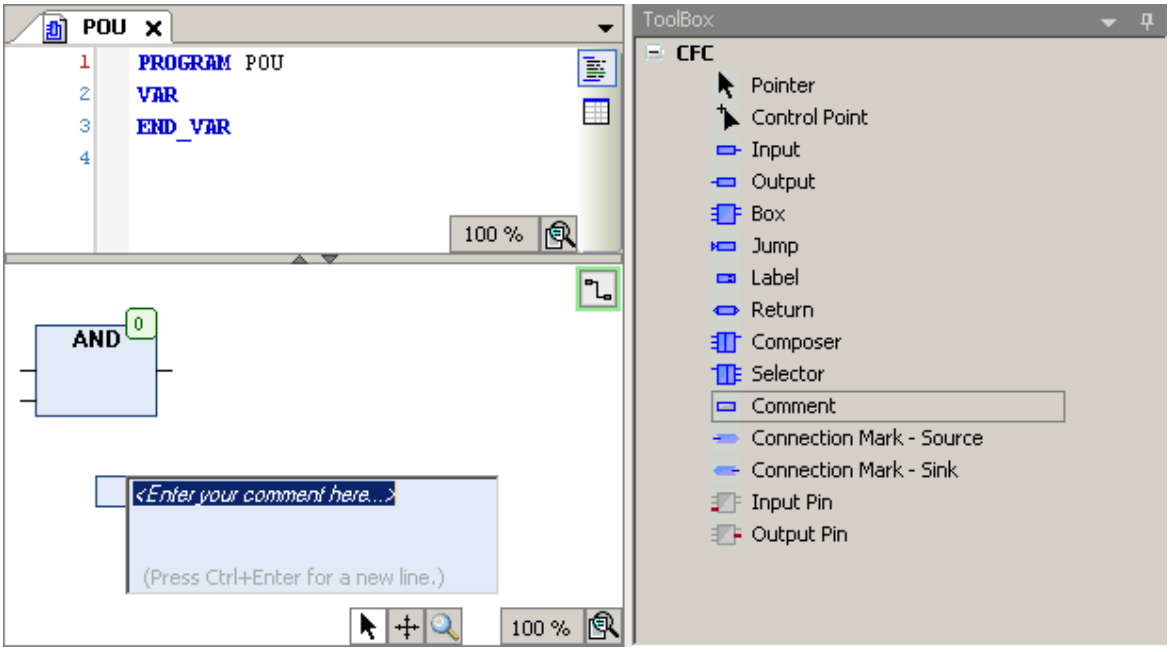
Each element has its own execution order. The number in the upper right corner of the element indicates the sequence in which the elements in a CFC network are executed in the online mode. The processing starts from the element with the lowest number, that is 0. Note that the sequence influences the result and are changed in certain cases.

You can set the execution order of each element using Set Execution Order and define the number.



Adding comments to a CFC program

In the ToolBox, select Comment and then drag and drop to the desired point in the program code area and enter the comment text.

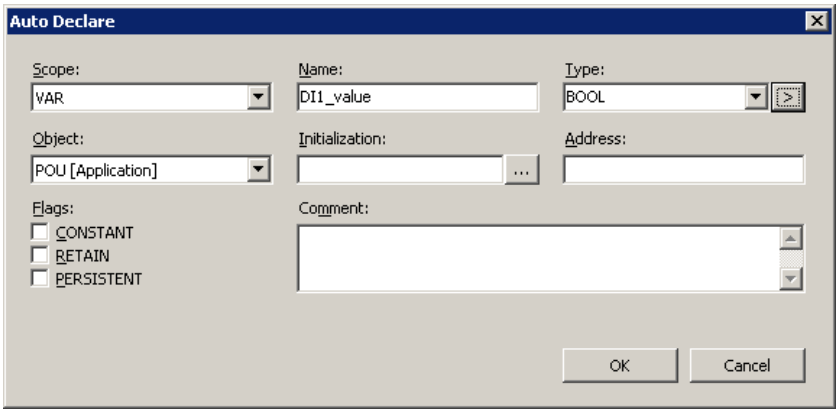


Declaring variables

To create a new variable, you can either declare it in the declaration part of the editor or use Auto declaration.

Depending on the type of the declaration view (textual or tabular) add a new variable by writing its properties to a new text row (textual view) or use the TAB button (tabular view). For changing between the views, see section *Writing a program code*.

1. In the program code area, select the required object.
2. In the Drive Application Builder main menu, go to **Edit** → **Browse** → **Auto Declare**. The Auto Declare dialog is displayed.



   If you enable the option to declare unknown variables automatically (**Tools** → **Options** → **SmartCoding**), the Auto Declare dialog opens every time you use an unknown variable in your program and you can declare the variable instantly.

3. Define the Scope, Name and Type of the variable (mandatory).
   - Scope defines the type of variable (global, input, output, etc.).
   - Name is a unique identifier of the variable and represents the purpose of the variable.
   - Type is the IEC data type of the variable.

Optionally, you can also define the Initialization value, Address, Comment or Flags for the variable.

Flags have the following meaning:

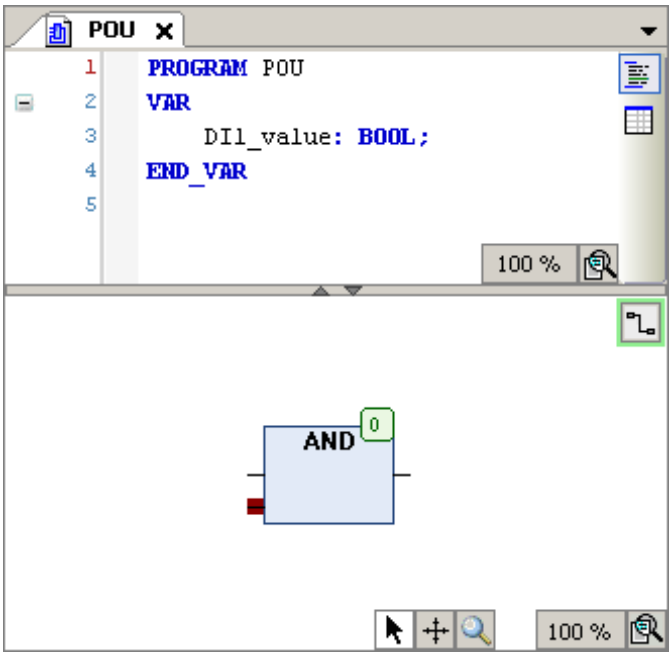
- CONSTANT means that the variable value cannot be changed and the variable maintains its initial value all the time.
- RETAIN keeps its value over reboot and warm reset.
- PERSISTENT is not supported.

Adding inputs and outputs

You can add inputs and outputs by selecting ToolBox elements. For further information, see section *Adding elements*.

Another way to add inputs and outputs straight to a block is to select a pin of a block and start typing the name of a variable.

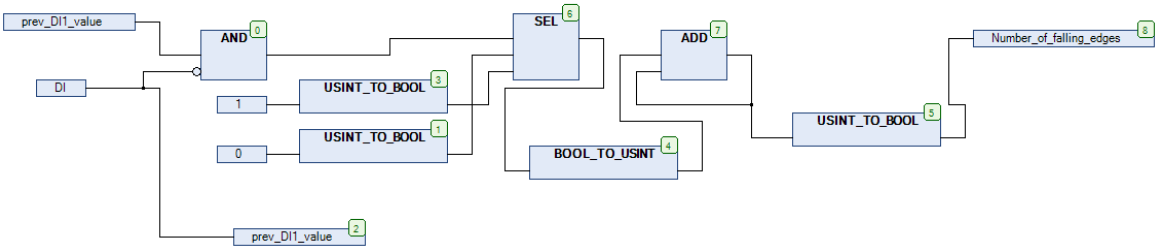
1. In the program code area, select the pin of the block.



2. Name the input or output by writing the variable name to the block or use input assistant as described in section *Declaring variables*.
3. To connect the input or output block to a pin, left-click the line connected to the block and drag it to a pin of another block.

CFC program

The below figure shows an example of CFC program.



The following local variables are required in the block scheme.

```
1  PROGRAM PLC_PRG
2  VAR
3
4      Number_of_falling_edges: BOOL;
5      prev_DI1_value: BOOL;// := False;
6      DI: BOOL;// := True;
7  END_VAR
8
```

During block scheme programming, the already created variables are displayed in the Input Assistant and new declarations are added to the variable declaration area.

For using the Input Assistant, see section *Adding elements (page 31)*.

Preparing a project for download

To prepare a project for downloading, follow these steps:

1. In the Drive Application Builder main menu, go to **Build** → **Build**.
2. Go to **View** → **Messages** to check that there are no errors or warnings.

Establishing online connection to the drive

The Drive Application Builder communication gateway handles communication between Drive Application Builder and the drive. The gateway is a software component that starts automatically at the powerup of the PC after installing Drive Application Builder.

Before starting with the communication setup, follow the pre-requisites listed below.

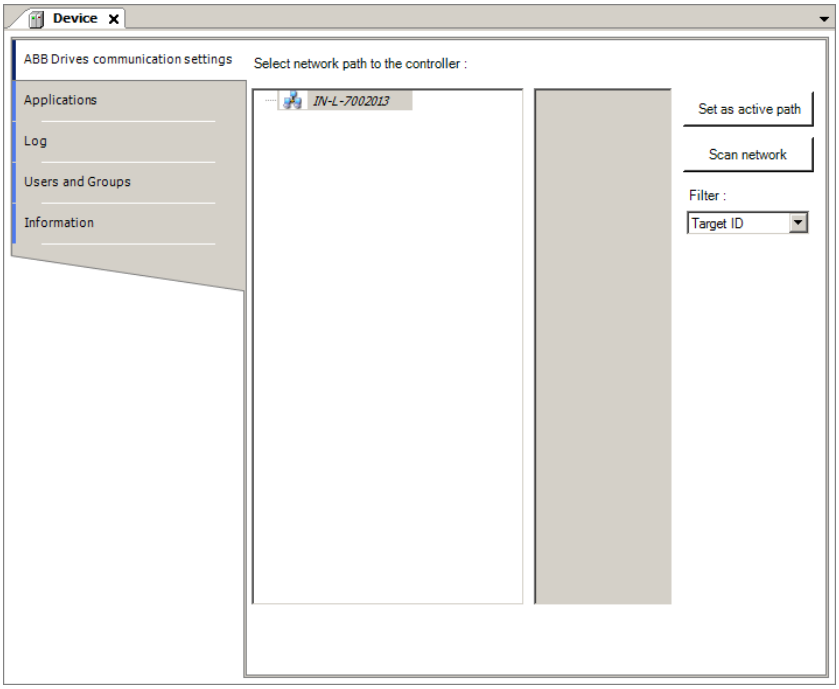
Pre-requisites:

1. Connect PC to a drive through USB port of the ACS-AP-x control panel using a standard USB data cable (USB Type A <-> USB Type Mini-B).
   For information on making the control panel to PC USB connection, see *ACS-AP-x control panel user's manual* (3AUA0000085685 [English]).
2. Make sure the ACS-AP-x USB driver is installed.
   For installation procedure, refer *Drive composer user's manual* (3AUA0000094606 [English]).
3. Make sure the drive has application programming license N8010. To check license information in Drive composer pro and in ACS-AP-x control panel, go to **System info** → **Licenses**.

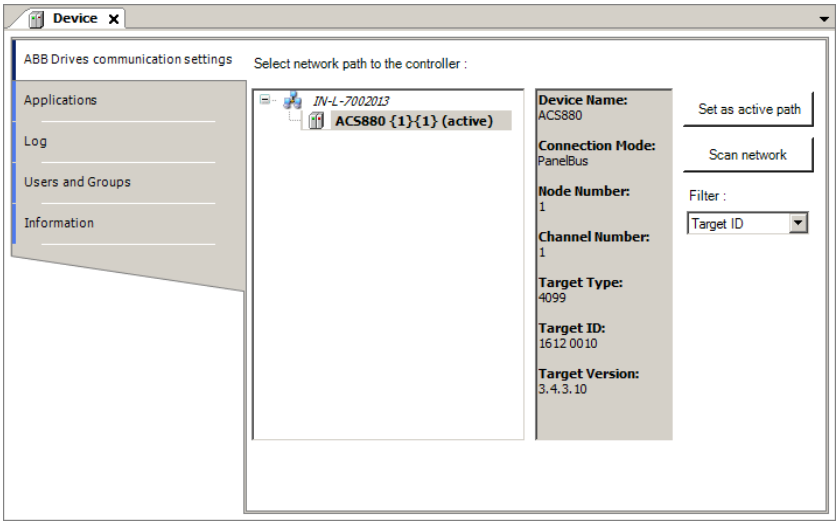
To establish online connection to the target drive after defining the device type, follow these steps:

1. In the Devices tree, double-click Device (ACS880_AINF_ZCU12_14_M_V3_5) and select **ABB Drives communication settings**.
   PC name is displayed by default.
2. Check that the USB cable is connected to the USB connector of the ACS-AP-x control panel and the drive is powered.

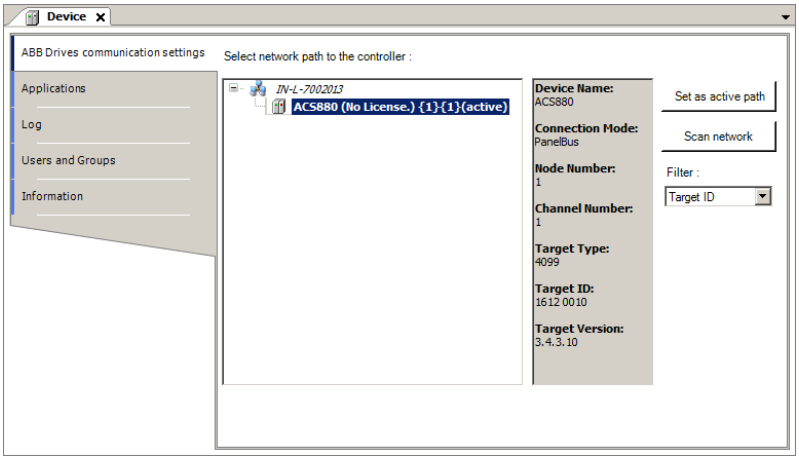
3. Double-click on first node (node with host name) or click **Scan network** to search the target device.
   Filter Target ID displays only devices that are of the same type as the device selected in the Devices window.



4. Double-click the device or click **Set as active Path**.



- If the drive has appropriate license code, the selected device is set as active path and is ready for downloading a program to the drive. See section *Downloading the program to the drive*.
- If the drive does not have the required license code, the selected device is displayed with no license.

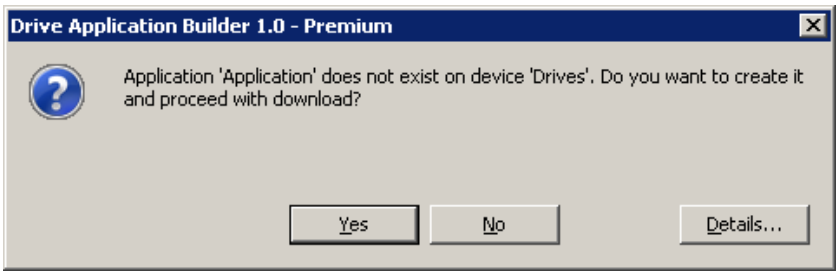

Device
ABB Drives communication settings
Applications
Log
Users and Groups
Information
Select network path to the controller :
IN-L-7002013
ACS880 (No License.) {1}{1}(active)
Device Name:
ACS880
Connection Mode:
PanelBus
Node Number:
1
Channel Number:
1
Target Type:
4099
Target ID:
1612 0010
Target Version:
3.4.3.10
Set as active path
Scan network
Filter :
Target ID

Downloading the program to the drive

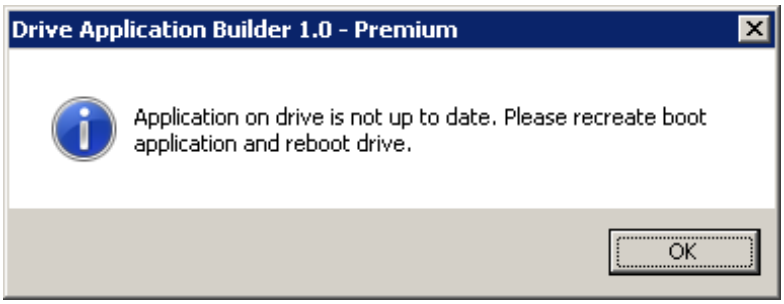
You can download and execute the written program to the drive after the project is ready for online communication with the drive. Check that the active path to the target device is defined in the communication settings. For more information, see section *Establishing online connection to the drive*.

In the Drive Application Builder main menu, go to **Online** → **Login**.

- If a program exists on the drive, the following dialog is displayed. Click **Yes**.

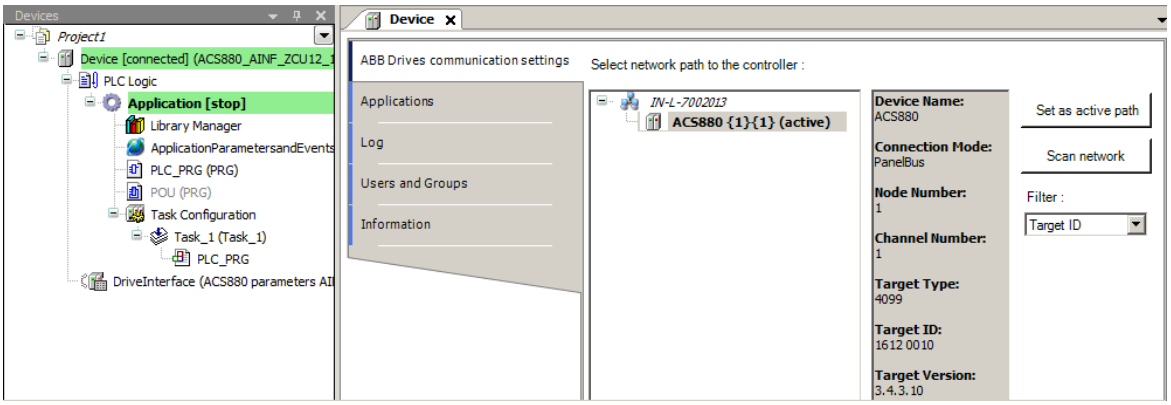


- If the application on the drive is not up to date, the following message is displayed. Click **OK** to recreate boot application and then reboot the drive.



After downloading the program, the background color of the device and application name in the Devices tree changes. The program is in stop mode and the status is shown in brackets [stop].

You can start the program by selecting **Start** in the **Debug** menu.



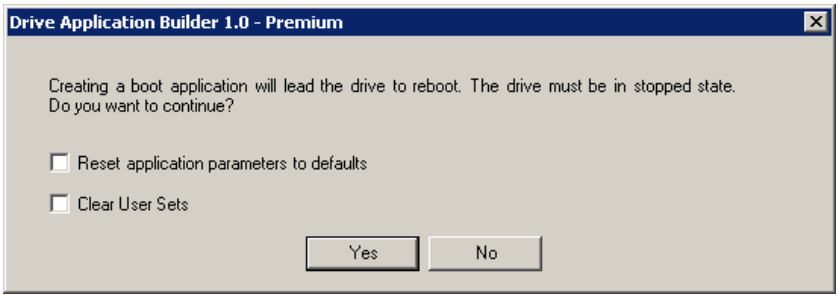
For more information on downloading a program, see section *Application download options (page 53)*.

Creating a boot project

The regular downloading moves the application program to the RAM memory of the drive. By creating a boot project, the application is copied to the non-volatile memory of the drive memory card and thus retains the application after power cycle or reboot. For more details, see section *Application download options (page 53)*.

To create a boot project, follow these steps:

1. In the Drive Application Builder main menu, go to **Online** → **Create boot application**. The following message is displayed. Click **Yes** to reboot the drive.



   The reset to default values is optional. If you select **Reset application parameters to defaults** option, the next boot resets all the application parameters to their default values. The previously set values are not restored from the permanent memory.

   Select this option when a new application is loaded to a drive or a reset origin has been performed or when application parameters of the new application are different from the previously loaded application.

   Note: It is recommended to select the **Reset application parameters to defaults** option whenever you load a new application to the drive or whenever you change the parameter definitions of the existing application (APEM).

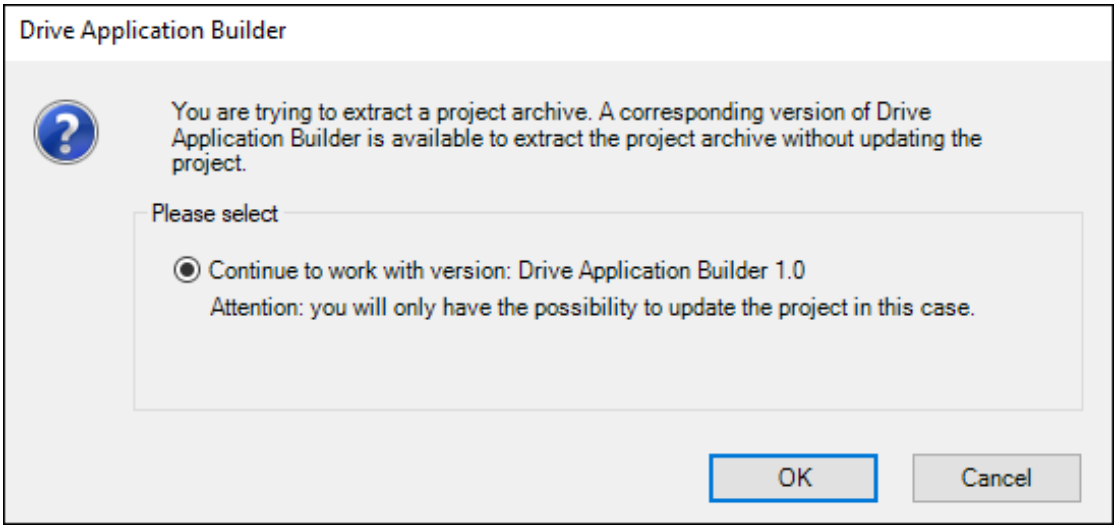
   If you select Clear User Sets option, user sets get cleared which are saved earlier using Drive Composer Pro.

   After creating the boot application, the status changes from STOP to RUN.

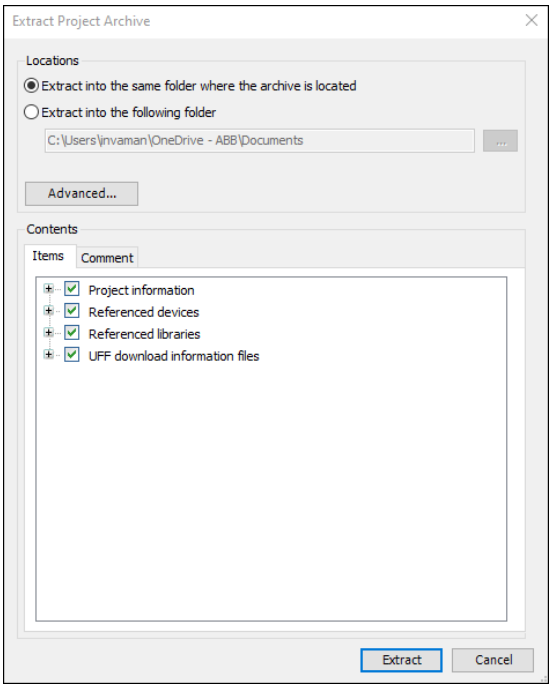
2. System prompts to save the boot application, click **Save**.

Opening a project archive

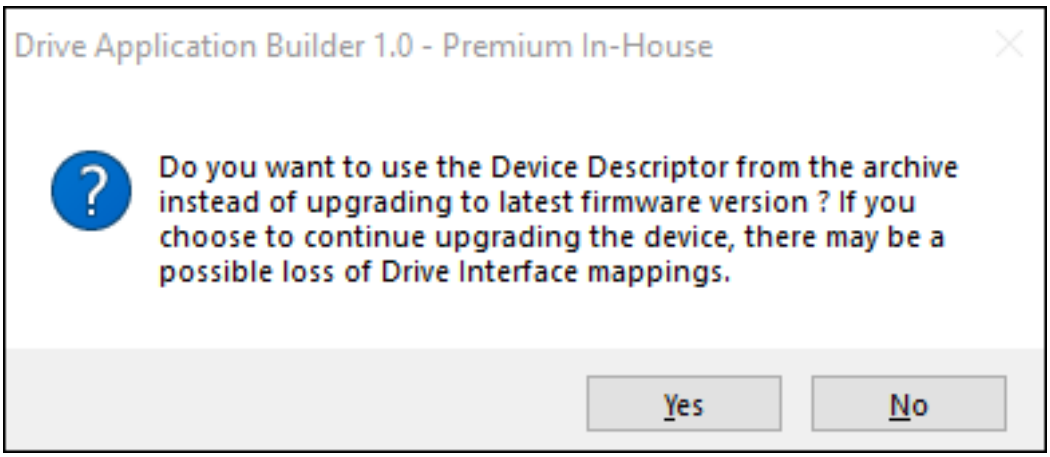
1. Launch **Drive Application Builder**.
2. In main menu, go to **File** → **Open Project**.
3. Navigate to a folder containing project archive and click **Open**.
   The below screen appears. Click **OK**.



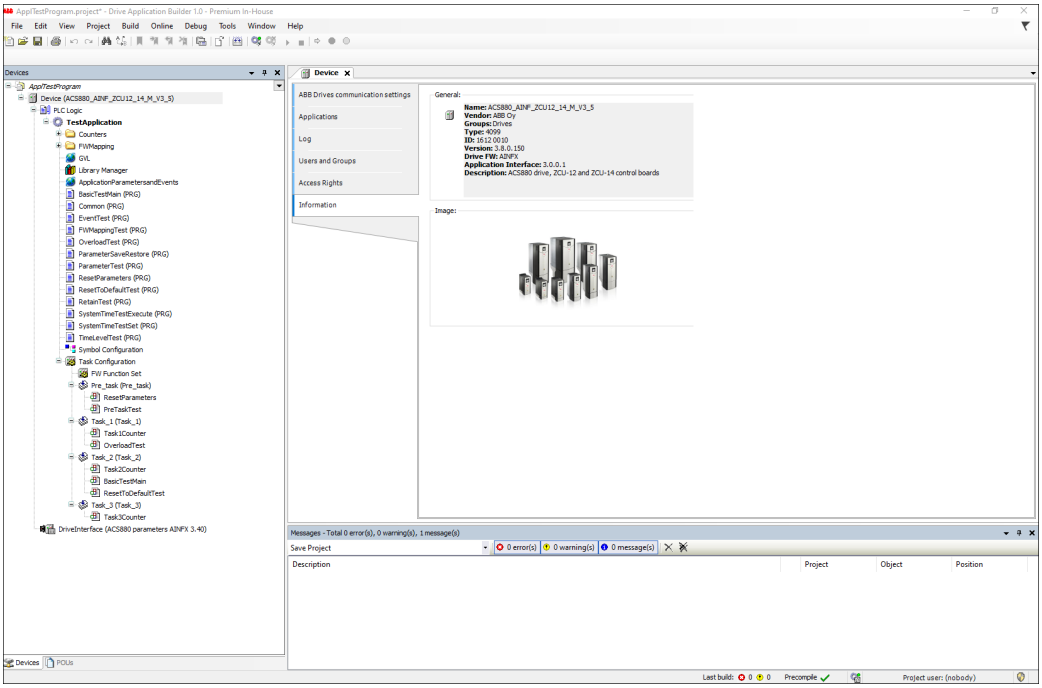
4. In Extract Project Archive dialog, choose the location to extract the archive and click **Extract**.



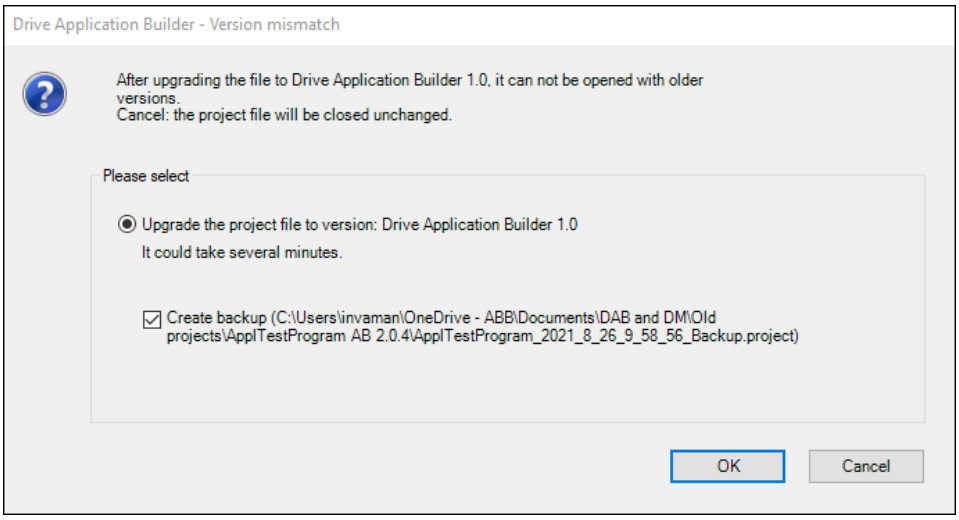
The below message appears.



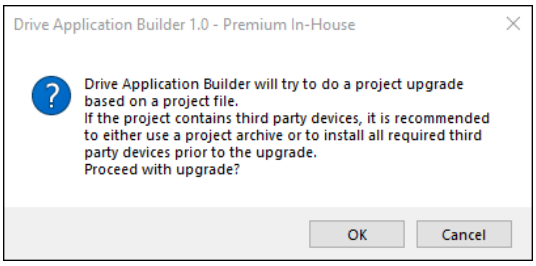
- Click **Yes** to load the project using device descriptions from archive. The project can be verified by checking the drive interface and device version.



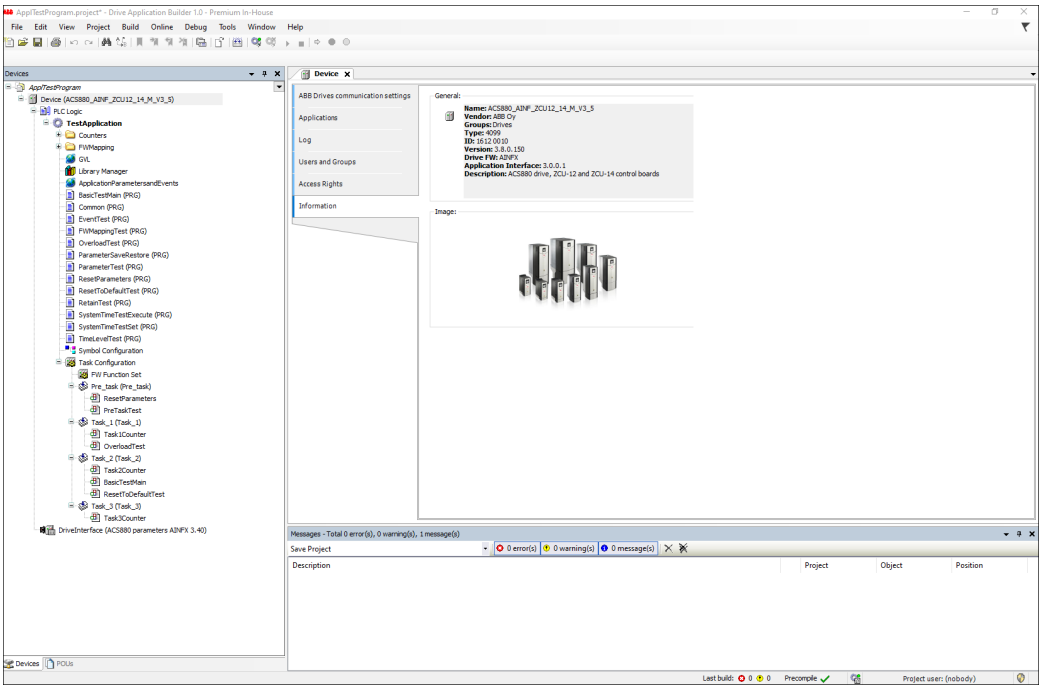
- Click **No** to use the latest version of device description available with Drive Application Builder. The below dialog appears. Click **OK**.



- If the project contains third party devices, the below message appears. Click **OK**.



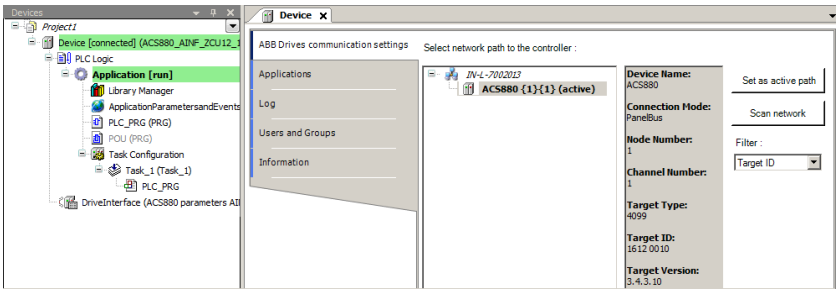
- If any parameter types missing due to project upgrade, a message appears. Click **OK**.
- If any compiler version upgrade is available, a message appears. Click **OK**.
- Verify the project update is completed by checking the device version and device interface firmware version.



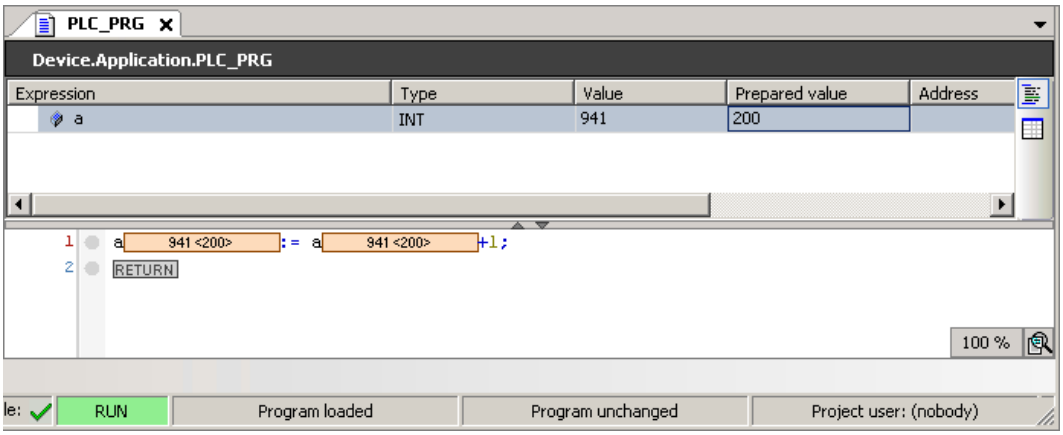
Executing the program

WARNING!
Do not debug or make changes to drive in the online mode or while the drive is running to avoid damage to the drive. Ignoring the instruction can cause physical injury or damage to the equipment.

1. In the Drive Application Builder main menu, go to **Debug** → **Start**.
   The application status changes to [run] and notifies that the program is executed successfully.



2. Double-click the cell in the Prepared value column and type a new value.
   - Press **Enter** to set or change a value of an existing variable.



3. In the Debug menu select the following:
   - **Write values** to apply the prepared value to the variable.
   - **Force values** to force the prepared value to the variable.
   - **Unforce values** to unforce a forced value.

   The variable value is changed. The current variable values are displayed in the Value column and in the source code at the variable.
4. In the Debug menu, click **Stop** to stop the drive.
5. In the Online menu, click **Logout** to logout.

5

Features

Contents of this chapter

This chapter describes the device handling information and features supported by Drive Application Builder.

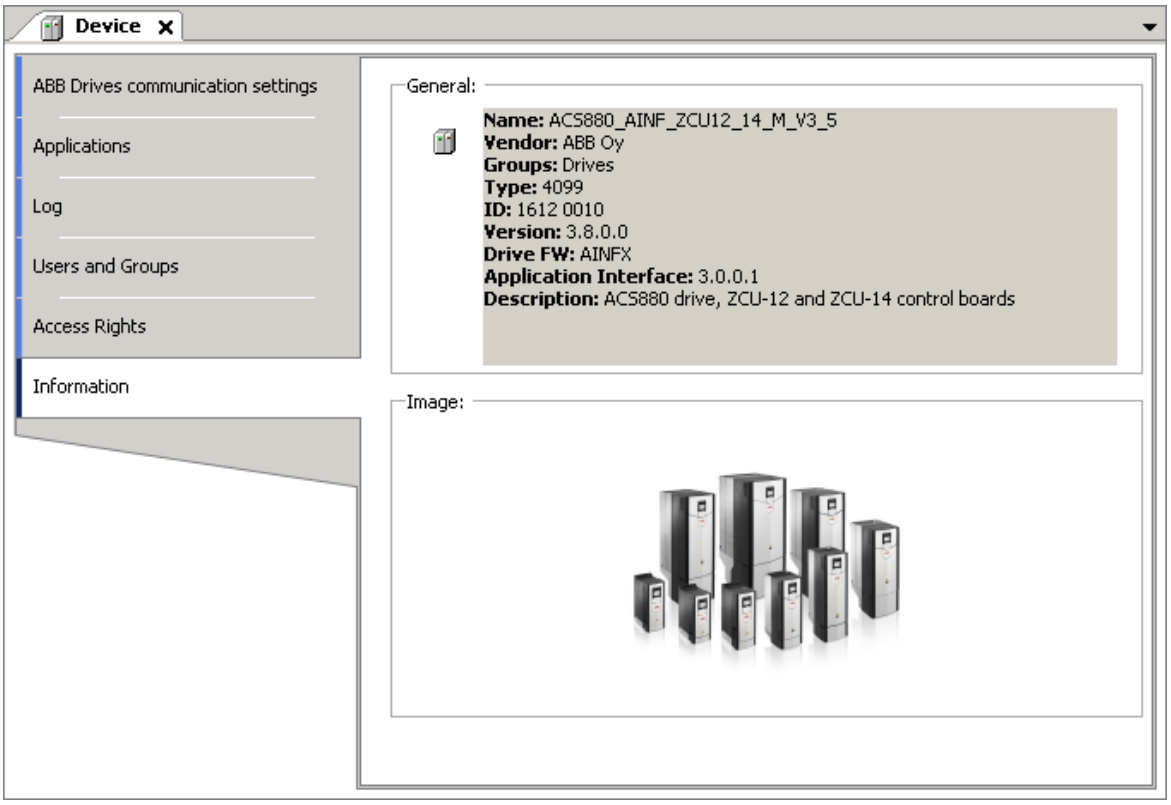
Device handling

In the application programming environment, devices represent hardware. The device description file contains information about the target device (drive) from the programming point of view like the device identifier, compiler type and memory size. The Drive Application Builder installation package installs the device description files automatically.

The device description may be updated later and a new file can be installed. The system monitors that a project with an incompatible device description file is not loaded to the drive.

Viewing device information

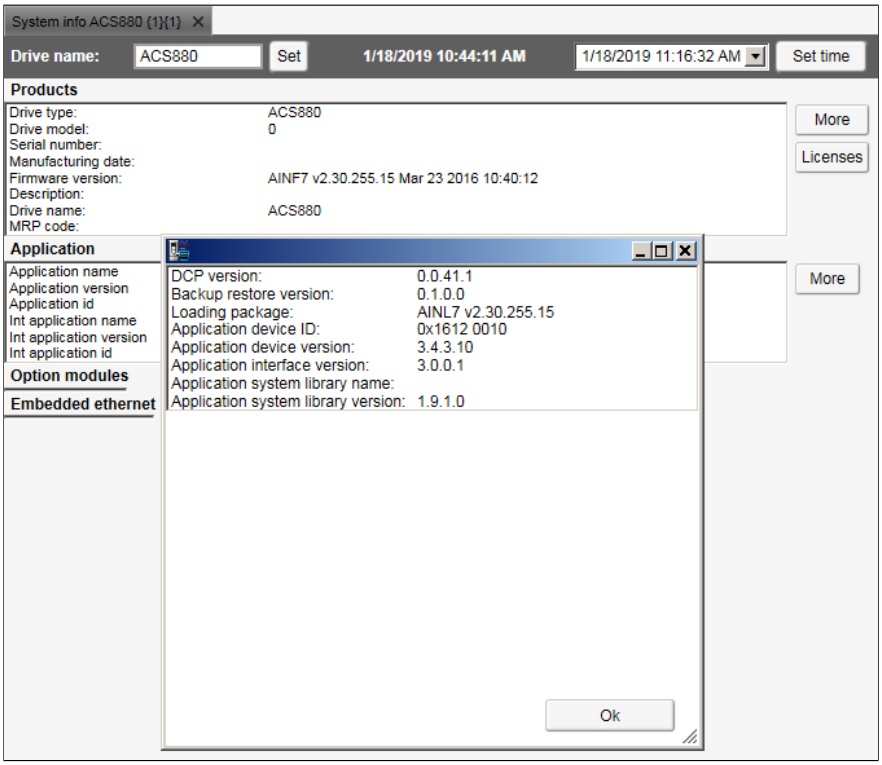
In the Drive Application Builder Devices tree, double-click on Device and go to Information tab to view the general information of a Device.



The Device ID, Drive FW name (AINFX) and application interface version must be identical in the project and drive target. In Drive composer pro, use the System info option to check that the drive target has the corresponding application interface version, device type and drive firmware name (displayed in parameter *7.04*).

You can also check if the drive target has the corresponding application interface version and device ID.

In Drive composer pro, go to **System info** → **Products** → **More**.



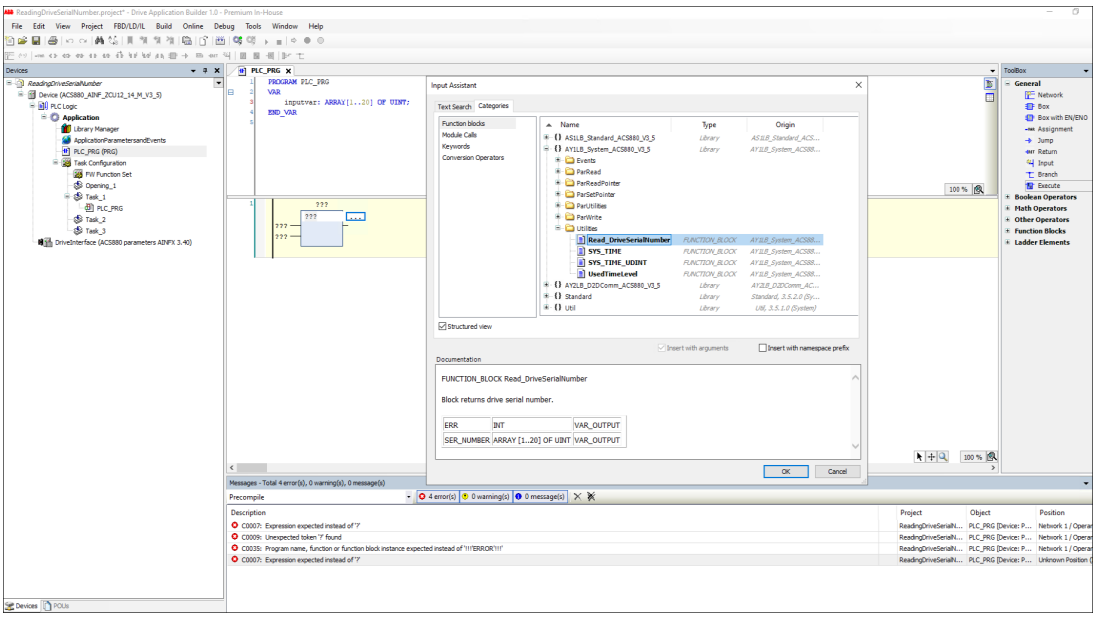
The name and version of the available system library is displayed. Make sure this information matches with the installed system library of the Drive Application Builder project.

For more information, see parameters *7.23* for Application name and parameter *7.24* for version in ACS880 FW.

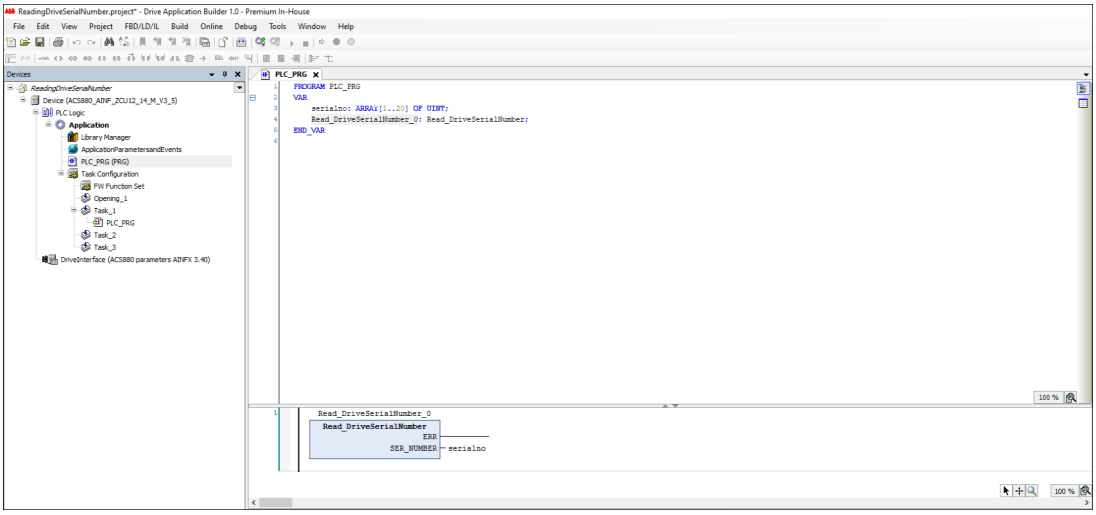
Viewing device serial number

To view device serial number, follow these steps:

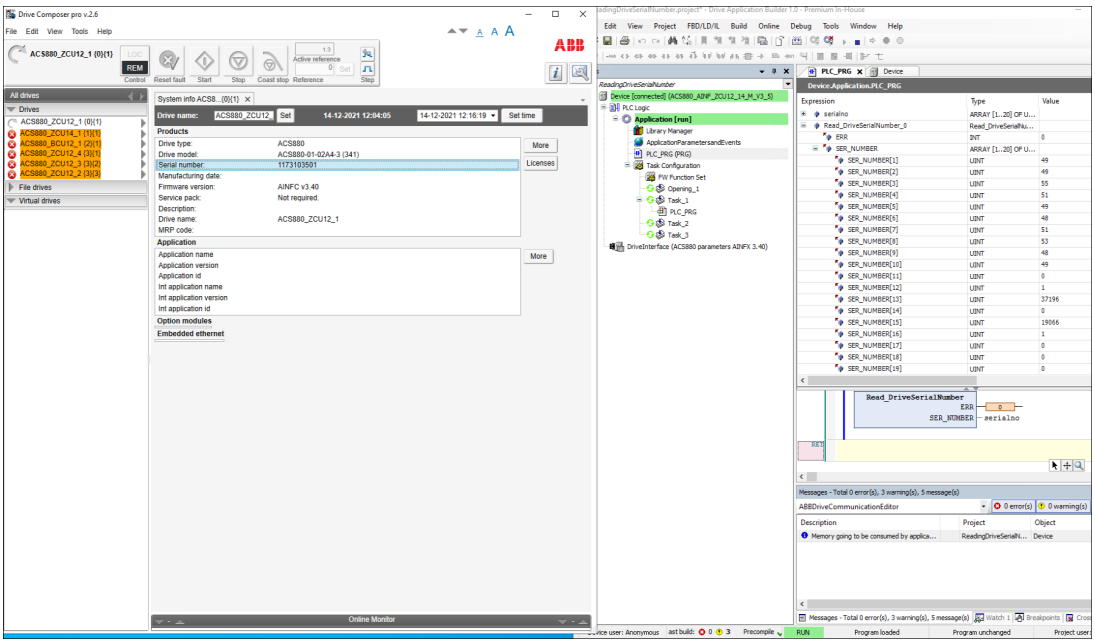
1. In Drive Application Builder, create a project with Ladder logic diagram. See section *Creating a new project (page 25)*.
2. Open PLC_PRG to create a variable of type ARRAY[1..20] of UINT.
3. Add an empty box to the ladder and select the box of type read_DriveSerialNumber.



4. Add a variable of type read_DriveSerialNumber to VAR and click **OK**.
   Check that the variable is added.
5. Assign variable of type ARRAY[1..20] of UINT to SER_NUM in the function block.



6. In the Drive Application Builder main menu, go to **Build** → **Build** and save the project.
7. Scan the drive device.
8. Create a boot application to download the application on the drive. See section *Creating a boot project (page 41)*.
   Make sure that the project is in RUN mode. Check the value of variables in drive composer pro and verify the serial number. The values are shown in ASCII value of the drive serial number.



Upgrading or adding a new device

You can upgrade or add a new device to the programming environment.

1. In the Drive Application Builder main menu, go to **Tools** → **Device Repository**.
   Device Repository dialog is displayed.
2. Click **Install** to select device description file.
3. In the Install Device Description window, browse and select the device description file (*.devdesc.xml*) in the file system.
   Now you can add a new device to projects or upgrade currently existing devices in the project.

Changing an existing device

You can change an existing device in Drive Application Builder project.

1. In the Drive Application Builder project, right-click on Device and select **Update objects** or in the main menu, go to **Project** → **Update project**.
   The Update objects dialog displays the available device types.
2. Select the required drive device and click **Update objects**.

Viewing software updates

In the Drive Application Builder start page, click Drive Application Builder to download **Drive Application Builder** update packages.

This link is a download center for Drive Application Builder. For example, you can find Drive Application Builder software, release note, Drive Application Builder update packages, and so on.

Program organization units (POU)

The POU types are:

- The program (PRG) can contain one or more inputs/outputs. A program can be called by another POU but cannot be called in a function (FUN). It is not possible to create program instances.
- The function (FUN) has always a return value and can have one or several inputs/outputs. The functions contain no internal state information.
- The function block (FB) has no return value but can contain one or more outputs as declared in the variable declaration area. A function block is always called using its instance and the instance are declared in a local or global scope.
  The created project can contain POUs with a specified implementation language. Each added POU has its own implementation language.

For detailed description of the POU types, see the *IEC programming environment user manual* and the *IEC 61131-3 open international standard*.

Data types

The ABB drives application program does not support some of the standard IEC data types like BYTE, SINT, USINT and STRING. The following list gives the standard IEC data types, sizes and ranges.

Data type	Size (bits)	Range	Supported by BCU-xx	Supported by ZCU-xx	Notes
BOOL	8/16*	0, 1 (FALSE, TRUE)	Yes	Yes	8 bit → BCU-xx 16 bit → ZCU-xx
SINT INT DINT LINT	8 16 32 64	-128...127 $-2^{15}...2^{15}-1$ $-2^{31}...2^{31}-1$ $-2^{63}...2^{63}-1$	Yes Yes Yes No	No Yes Yes Yes	
USINT UINT UDINT ULINT	8 16 32 64	0...255 0...65535 $0...2^{32}$ $0...2^{64}$	Yes Yes Yes No	No Yes Yes Yes	
BYTE WORD DWORD LWORD	8 16 32 64	0...255 0...65535 $0...2^{32}-1$ $0...2^{64}-1$	Yes Yes Yes No	No Yes Yes Yes	
REAL LREAL	32 64	$-1.2*10^{-38}...3.4*10^{38}$ $-2.3*10^{-308}...1.7*10^{308}$	Yes Yes	Yes Yes	Slow. Do not use.
TIME LTIME	32 64	0 ms...1193h2m47s295ms 0 ns...~213503d	Yes Yes	Yes Yes	
TOD	32	00:00:00...23:59:59	Yes	Yes	
DATE DT	32 64	01.01.1970...~06.02.2106 01.01.1970 00:00...~06.02.2106 00:00	Yes Yes	Yes Yes	
STRING[xx]		0...255 characters	Yes	No	
WSTRING[xx]		0...32767 characters	Yes	Yes	

Drive application programming license

The drive application programming license N8010 is required to download and execute the program code on ABB drives. To check license information in Drive composer pro or in ACS-AP-x control panel, go to **System info** → **Licenses**. If the required license code is not available, contact your local ABB representative.

Application download options

Before executing an application in the drive, download the application to the drive memory. After downloading, the application software is embedded in the firmware of the drive and has access to system resources.

Note: It is not recommended to download a program to the RAM memory when the drive is in RUN mode. The drive must be in STOP mode and Start inhibits must be possible to set.

Before downloading, make sure that there is no fieldbus device, M/F-link or D2D-link connected to the drive. Drive composer is not running data monitoring or back-up/restore at the same time.

There are two different download options:

- **Download** - This is a regular download method that copies the compiled application to the drive RAM memory. As a result, it is possible to execute the application, but after a power cycle or reboot the memory is erased. This download method does not alter an application that is located in the drive boot memory (ZMU) and the original application is available for use after a reboot.
- **Create boot application** - This download method copies the application to the non-volatile memory of the drive memory card. This way the application remains intact after a power cycle or reboot. You should be logged into the drive to perform this operation. Features that can work only after restarting the drive should be downloaded with this method.

  Create boot application command (**Online** → **Create boot application**) also includes booting the drive. Rebooting stops the execution of the complete drive firmware for some time. For this reason, it is allowed only when the drive is stopped and start inhibition is granted to the **Drive Application Builder**.

Note:

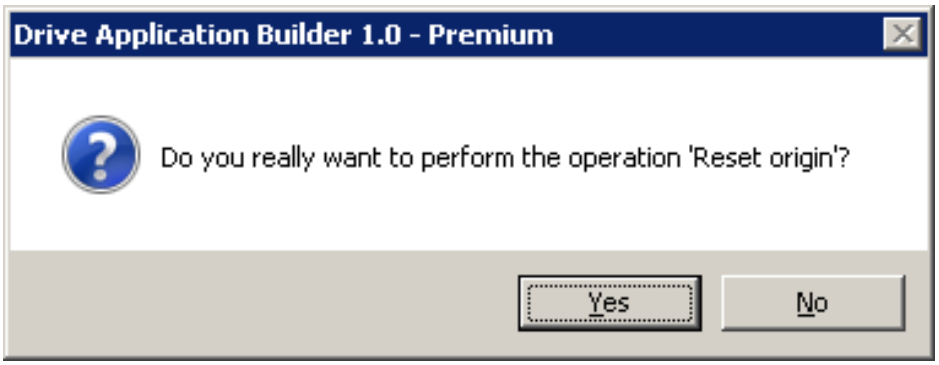
- Firmware parameter mapping, task configuration, application parameters and event configuration are activated only after the boot application is loaded and the drive is booted.
- Start inhibition is not granted if the drive is running, disabled (DIL, Safety function active) or faulted. Make sure that these conditions do not exist before downloading the program.

Removing the application from the target

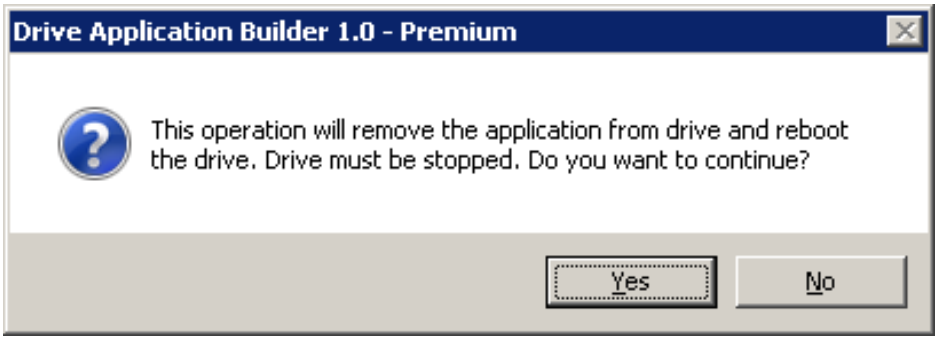
Drive Application Builder application allows you to remove application from the target. You can use Reset option if the application includes many changes like application parameter changes or the application is replaced by another application. If the target already includes an application, use the Reset origin selection in the Online menu before downloading a new application.

This command removes (clears all) old application from the target and all the application related references. Use this command at least once before the final version of application is loaded. The command can be used only in the online mode. For further information on Reset options, see section *Reset options*.

When you are prompted with the following message, click **Yes**.



After you initiate the Reset origin option, the following message is displayed. Click **Yes**. The command is executed only if Drive Application Builder receives the permission from the drive.



Retain variables

Retain variables includes the RETAIN flag used to retain values throughout the drive reboot and warm reset. A cold reset sets the retain variable to its initial value. The values of retain variables are cyclically stored in the flash memory of the drive and restored to the stored value after restarting the program. The retain variables are stored in a separate 256-byte memory area which defines the limits of their amount.

WARNING!
In a function block, do not declare a local variable as RETAIN because the complete instance of the function block is saved in the retain memory area and this large function block instance can lead to running out of memory space.

In firmware version 2.6 and later, the power control board works with the parameter settings:

- If parameter *95.04* = Internal 24V, retain values are saved immediately at the time the drive loses power, meaning it is not cyclical.
- If parameter *95.04* = External 24V, retain values are saved at periodic intervals of 3 minutes. So the recovered variable may not be the recent value.

Note: Declaring a local variable in a function as RETAIN has no effect and the variable is not saved in the retain memory area.

The existing retain variables cannot be linked to application parameters.

Task configuration

The task configuration object handles call configuration of the programs. A task is a project unit that defines which program is called in the project and when it is called. The project can have more than one task with different time levels.

There are two types of tasks:

- Cyclic task (Task_1, Task_2 and Task_3) - The task is processed cyclically according to the task cycle time interval. The following table lists the time intervals available for cyclic application programs. The highest priority is given to the task with the shortest execution interval.

Task	Time interval
Task_1	1 … 100 ms
Task_2	10 … 100 ms
Task_3	100 … 1000 ms

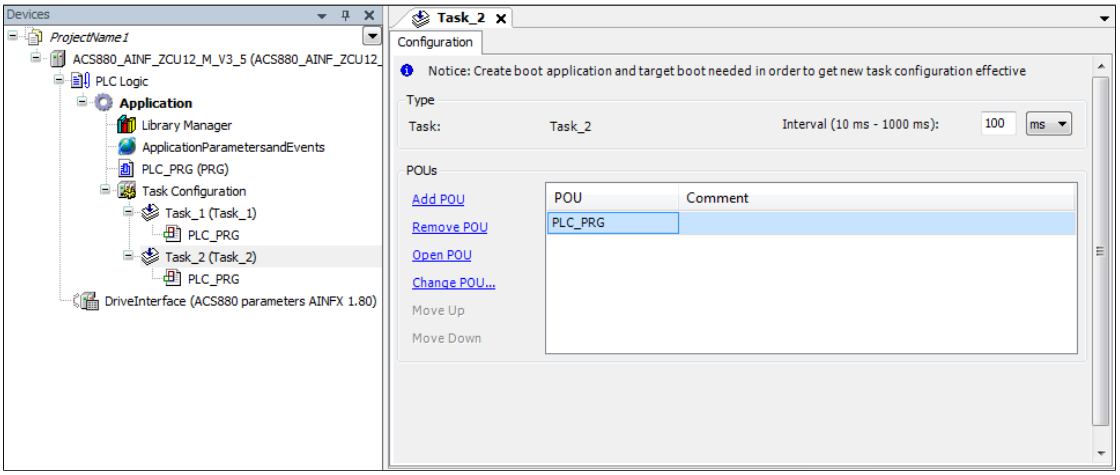
- Pre_task - The task is executed only once at start-up of the application program. The feature is useful for one time initialization. The POUs (blocks) assigned into this task are executed before starting the cyclic tasks.

Note: The application program consists of specific allocation of CPU resources. If the limit exceeds, the drive trips to task overflow fault. For details, see *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].

Adding tasks

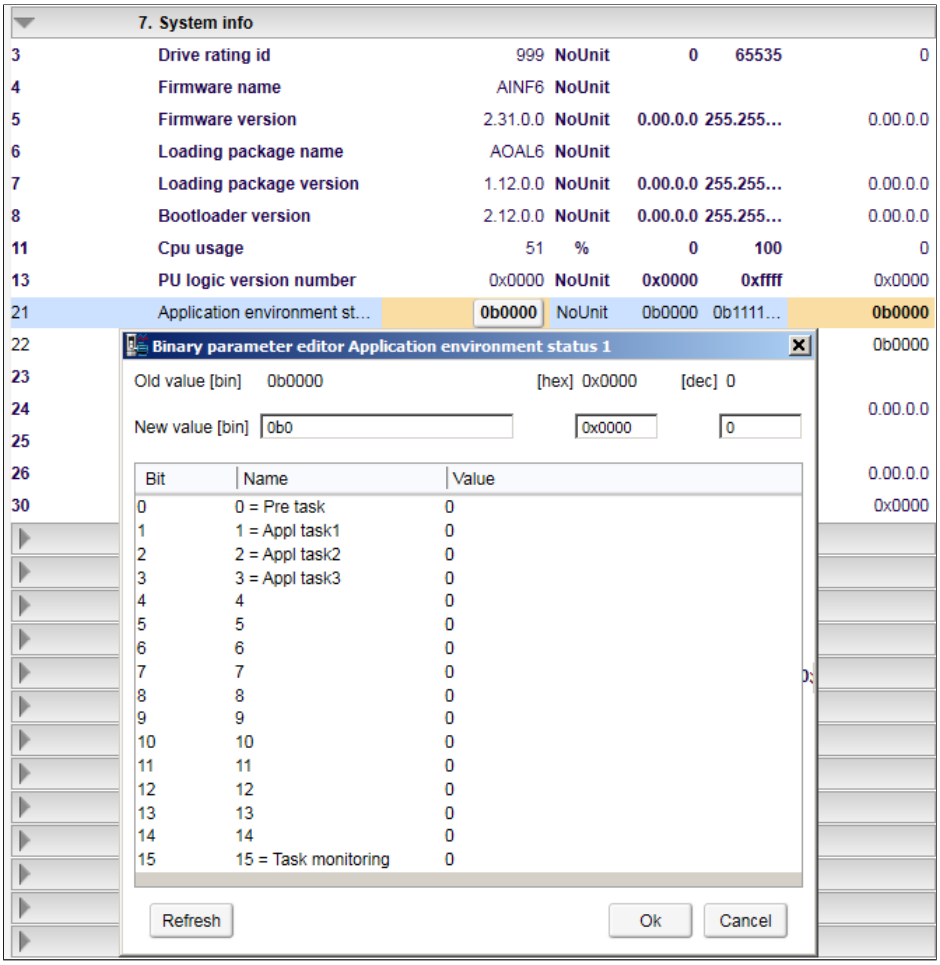
To add tasks to Task Configuration, follow these steps:

1. In the Devices tree, right-click **Task Configuration** and select **Add Object**.
2. Select the Task and click **Add object**.
3. Select the Task in drop-down list and click **Add**.
   The selected tasks are added in the Task Configuration object.
4. Click **Add POU** in the newly added Task_2 screen.
5. In the Input Assistant dialog, click Categories tab and then select **PLC_PRG** and click **OK**.
   PLC_PRG is added to Task_2. Drag and drop PLC_PRG to Task Configuration object.


Devices
ProjectName1
ACS880_AINF_ZCU12_M_V3_5 (ACS880_AINF_ZCU12_
PLC Logic
Application
Library Manager
ApplicationParametersandEvents
PLC_PRG (PRG)
Task Configuration
Task_1 (Task_1)
PLC_PRG
Task_2 (Task_2)
PLC_PRG
DriveInterface (ACS880 parameters AINFX 1.80)
Task_2
Configuration
Notice: Create boot application and target boot needed in order to get new task configuration effective
Type
Task:
Task_2
Interval (10 ms - 1000 ms):
100
ms
POUs
Add POU
Remove POU
Open POU
Change POU...
Move Up
Move Down
POU
Comment
PLC_PRG

Monitoring tasks

Before adding the tasks for monitoring in Drive Application Builder, check parameter *7.21 Application environment status* in Drive composer pro.



The parameter bits 7.21.0, 7.21.1, 7.21.2, and 7.21.3 are used to monitor the application task related execution. To check the continuous execution of tasks, write the specific task bit to 0. The executing task bits are updated to 1, except the Pre task, which executes only once.

The calculation of tasks execution cycle (duration) is disabled by default. To view the tasks execution monitoring in Drive Application Builder, change Bit 15 = Task monitoring to high.

To add task monitoring view in Drive Application Builder, follow these steps:

1. In the Devices tree, double click **Task Configuration**.
2. Click **Monitor** tab to check the status report of available tasks.
   The status report of available tasks appears. The values in the task monitoring view are updated only after setting the parameter 7.21.15 to high in Drive composer pro. The setting is configured again after the power cycle or boot or control board.

You can evaluate the total (task 1-3) CPU load using the parameters *7.40 IEC Application Cpu usage peak* and *7.41 IEC Application Cpu load average*. For parameter descriptions, see *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].

Uploading and downloading source code

Optionally, the source code of the project can be saved in the drive. This feature is located in Drive Application Builder main menu **Online** → **Source download to connected device** or in Device tree, right-click on drive device and click **Source download to drive** and it ensures that the files are easy to obtain if needed.

You can retrieve the saved source code from the drive to a new project using **File** → **Source upload from drive** option available in Drive Application Builder main menu and then scan and select the drive.

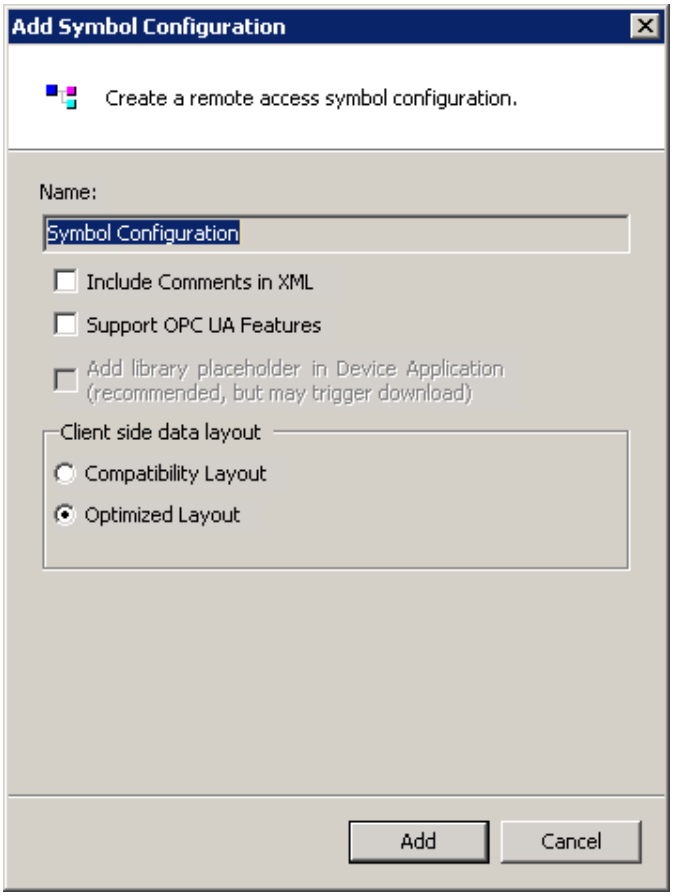
The size of the source code is limited to 500 KB. Check the archiving option to minimize the source code size (**File** → **Project Archive** → **Save/Send Archive…**). Note that referenced devices and libraries are needed, the rest is optional.

Note: If the source code is saved on the ZMU memory unit, you can retrieve the program with another PC without the authors consent unless the project is password protected.

Adding symbol configuration

To add symbol configuration in Drive Application Builder project, follow these steps:

1. In the Devices tree, right-click Application and select **Add object**.
2. Select Symbol configuration and click **Add object**.
3. In the Add Symbol configuration dialog, click **Add**.



Symbol configuration object is added to the project.

After adding Symbol configuration object to the project, the IEC variable to symbol data is loaded into the drive during the create boot application download. See section *Application download options (page 53)*. The feature provides Drive composer pro access to the application variables which is used for graphical monitoring and debugging.

Debugging and online changes

The following debugging features and variable forcing are supported:

- Start/stop program execution
- Setting breakpoints
- Stepping code line by line or by function
- Forcing variables (constant setting of variable values)
- Writing variables (single setting of variable values)

Note: Online changes of the program code are not supported.

WARNING!
Ignoring the following instruction can cause physical injury or damage to the equipment.

Do not set breakpoints and force variables on a running drive that is connected to motor.

Safe debugging

Avoid the following actions when debugging the application program of a running drive connected to motor in the online mode:

- stopping the application program
- setting breakpoints to the application program
- forcing variable values
- assigning values to outputs
- changing the values of a local variable in function blocks
- assigning invalid input values

Breakpoints stop the entire application, instead of just the task that has the currently active breakpoint.

Reset options

You can reset the application, using the reset selections in the Online mode.

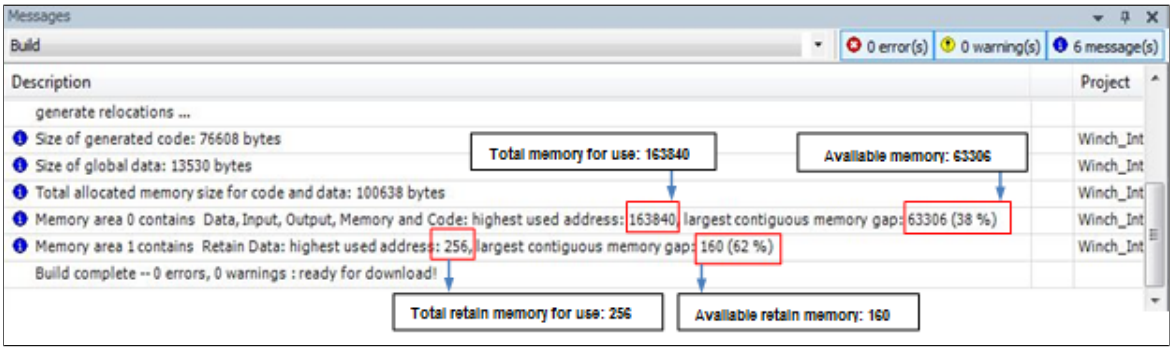
1. In the Devices tree, select the Application.
2. In the main menu, click **Online** and select the desired reset method.
   - **Reset warm** reset all variables of the currently active application to their initial values (except retain and persistent variables). In case of specific initial values, the variables are reset exactly to those specific values.
   - **Reset cold** reset all variables (normal and retain) of the currently active application to their initial values.
   - **Reset origin** erase the application downloaded to the drive from the RAM and the memory unit (Boot application). In case of specific initial values, variables are reset to those specific values. Drive firmware parameter mappings, user-defined parameters and events are also removed. Finally the drive is restarted.

Memory limits

You can remove the temporary code sections from the program using **Build** → **Clean** or **Clean all** options available in Drive Application Builder main menu.

The memory area 0 is assigned for code and data. Memory area 1 is assigned for retain variables.

The below example shows an actual allocation in the build report.



Note: To optimize the memory consumption, avoid using function blocks and unnecessary variable definitions.

CPU limitation

The maximum execution load of the application is limited to 5 to 15% depending on the drive type. To know the actual load limit, contact your local ABB representative.

You can monitor the CPU load by checking the application load with parameter *7.11 Cpu usage*. To know the load difference, compare CPU usage values with and without the application. Make sure that the difference value is not greater than the value limit. If the application exceeds the limit, the drive trips to the task overload fault 6481. The fault is registered to the event log of the drive and the fault-specific AUX code indicates the overloaded tasks (10 = task 1, 11 = task 2 and 12= task 3).

You can evaluate the total (task 1-3) CPU load using the parameters *7.40 IEC Application Cpu usage peak* and *7.41 IEC Application Cpu load average*. For parameter descriptions, see *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].

Perform CPU load tests to make sure that the drive is capable of adequately running the application. Enable the required drive functions during the execution of the application. For example, motor control, communication modules, encoders, and so on.

Application loading package

This feature allows the user to create a loading package containing of an application program for ACS880 drive. To build a loading package when the tool is connected online to the drive, use the Drive Application Builder command **Create Boot Application**.

You can also create offline application loading package file without firmware version limit using premium license.

Note: To create loading package with or without firmware restrictions in offline, you must have premium license.

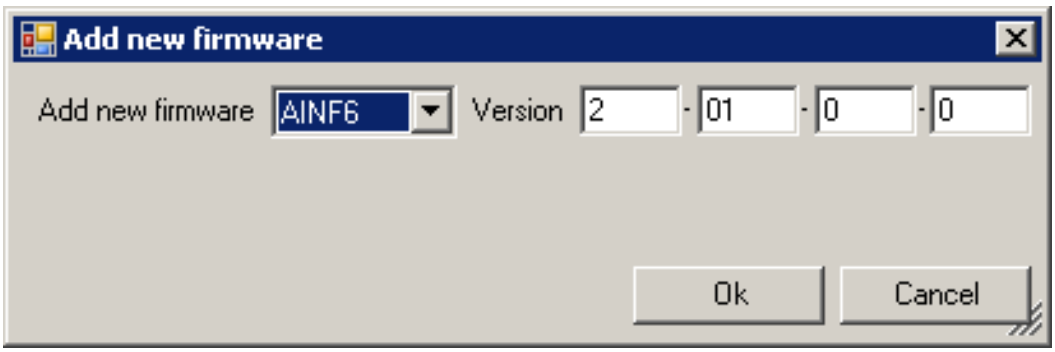
Place the file to the corresponding project folder with the file name *<project_name>_<device>_<application>.lp*. Load the application loading packing using the Drive loader tool.

Note: Application loading package functionality supports from AINFX 2.01 firmware version onwards.

Before loading the package, Drive loader tool checks for the correct actual drive type and firmware version to load the package. It also checks for the correct drive application programming interface and the active programming license (N8010) in the target drive.

To include symbol data and source code to application wrap file and loading package using Drive Application Builder, follow these steps:

1. In the main menu of Drive Application Builder, go to **Project** → **Project Settings**. Project settings dialog is displayed.
2. Click **Application loading settings** and select the desired options.
   - Click add icon to add new firmware.
   - Enter the firmware details and click **Ok**.



The added firmware is displayed in the Application loading settings.

Note: Make sure that the application is working correctly with the added firmware.

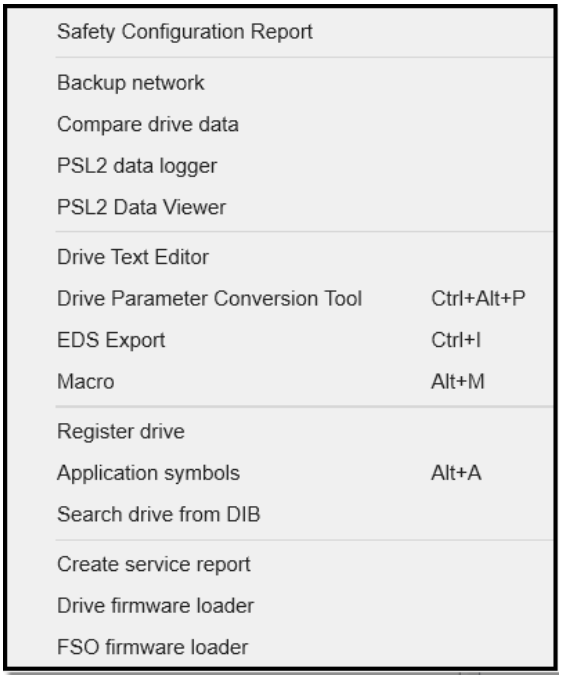
It is also possible to add more supported firmware versions to the application loading package.

Loading firmware to a drive

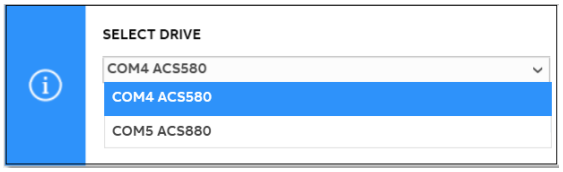
To load the drive firmware, make sure the drive is connected to Drive Composer via USB cable. For more information, see *Drive composer user's manual* (3AUA0000094606 [English]).

1. Power up the drive. Make sure that the motor is not running (inverter) or the unit is not operating (supply unit). If drive is connected via USB cable, Drive Composer automatically discovers the drive.

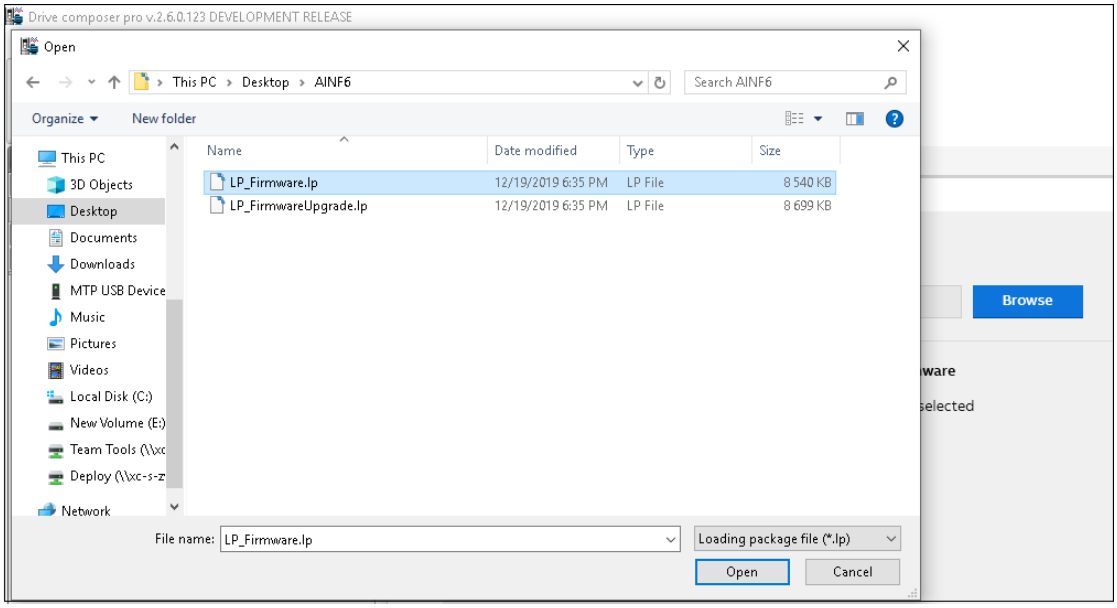
2. In main menu, click **Tools** → **Drive firmware loader**.



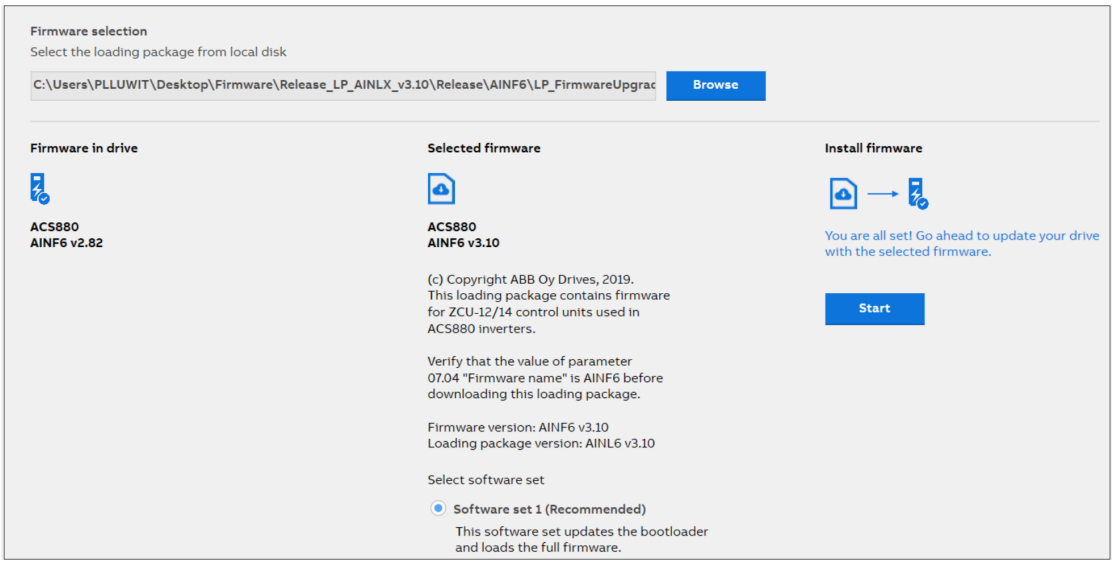
3. If multiple drives are connected, from the selection list, select the drive to load the firmware and click **OK**.



4. Click **Browse** and select the loading package file.

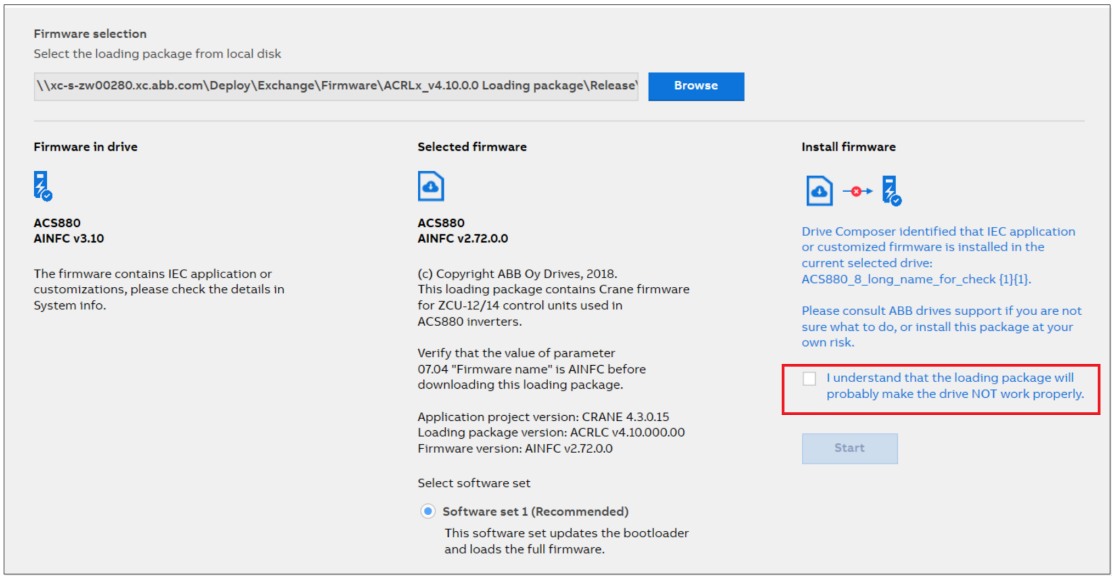


5. If applicable, select the software set.



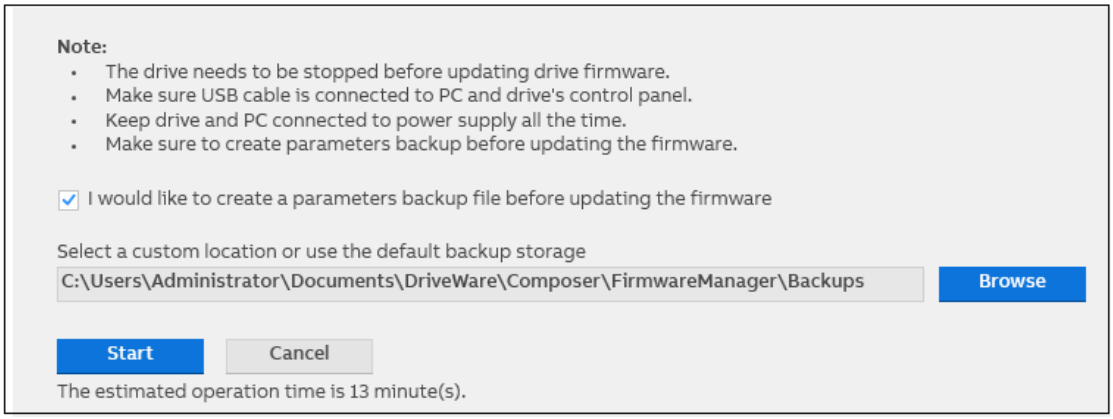
6. If applicable, select the acceptance statement check box. This appears when you try to downgrade the firmware, load possibly incompatible firmware or if you are using customized firmware.

   Note: If you try to load a corrupted or invalid loading package file, the application displays an error message.

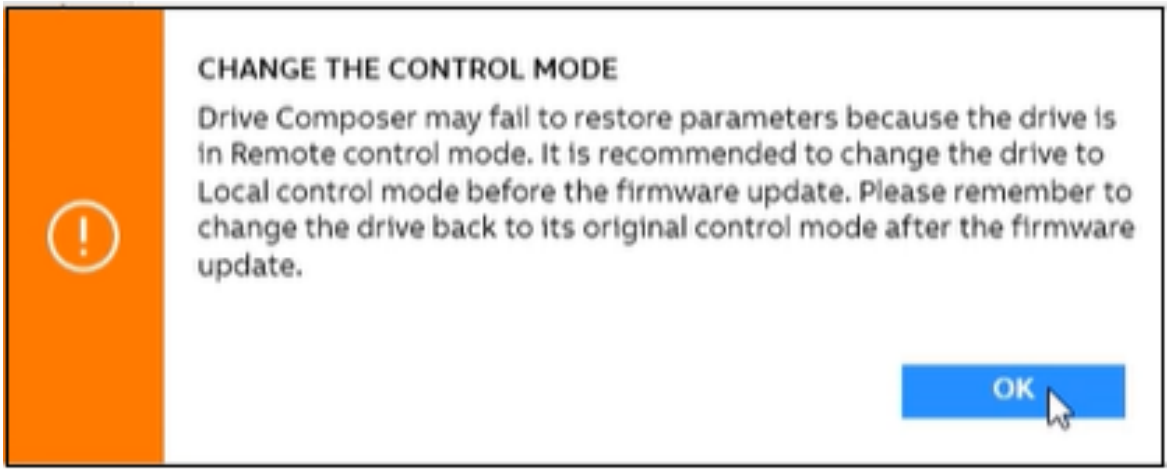


7. Click **Start**.

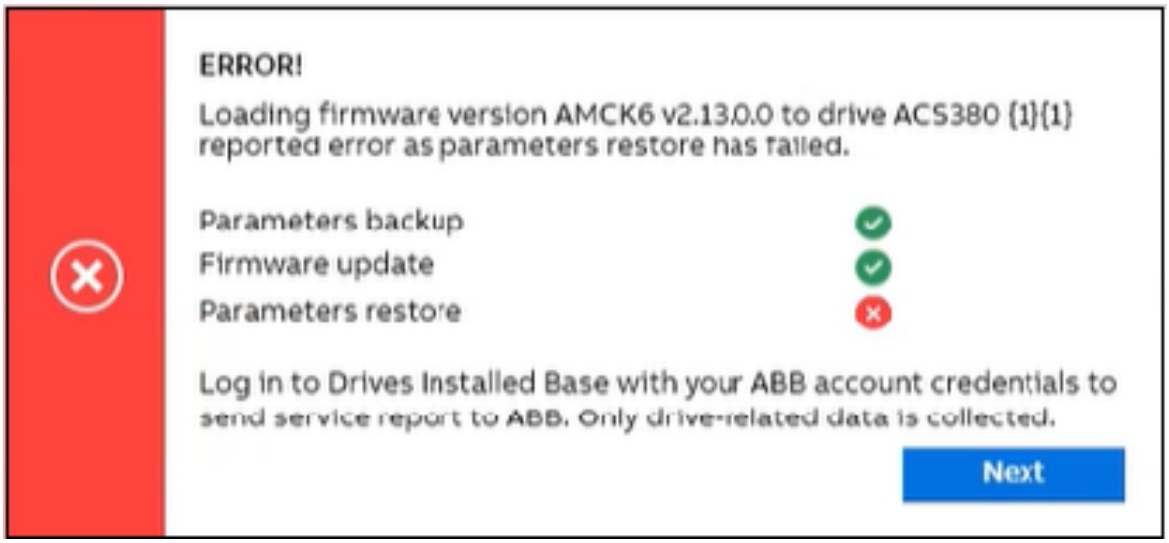
8. Select the check box for I would like to create a parameter backup file before updating the firmware, and select the location to save the file.



9. In the drives that do not support parameter restore in remote control mode, a warning dialog box appears if the drive is in remote control mode. Click **OK** and change the drive control mode to **Local**.

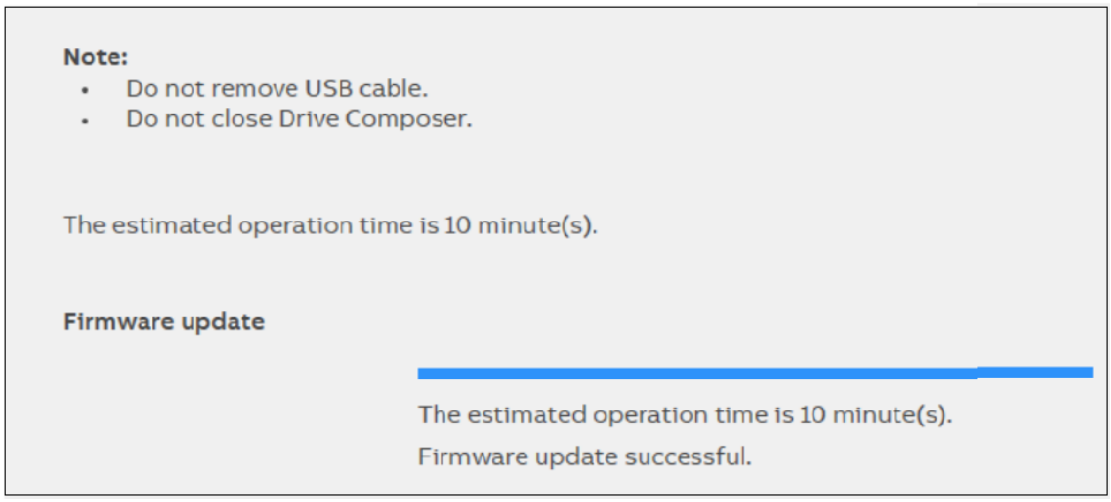


If you proceed the operation with remote control mode, the following error message appears.

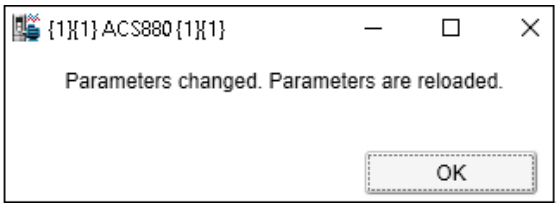


10. Click **Start**.

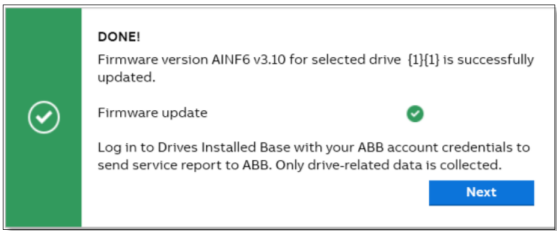
11. Wait for the firmware to be loaded. The backup time can be more if you selected the parameter backup option.



12. The message Parameters changed. Parameters are reloaded appears. Click **OK**.



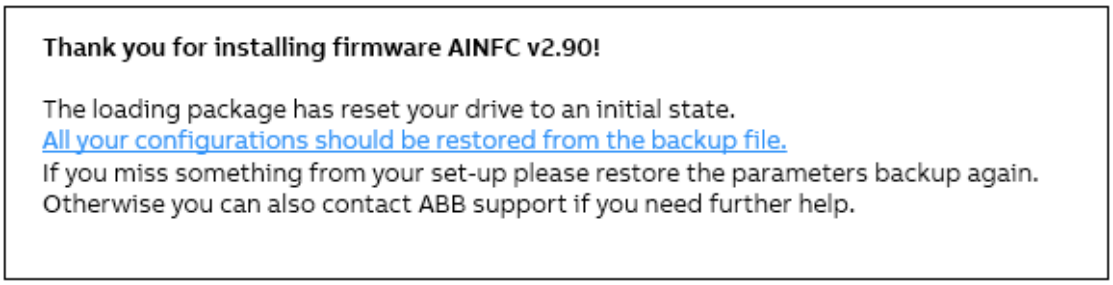
13. In the DONE! dialog box, click **Next**.



14. Optional (only with access to DIB): Enter the service report information.
    For more information on how to create online service report, see *Drive composer user's manual* (3AUA0000094606 [English]).

    Note: You need access to Drive installed base to send the service report.

15. Optional (only with access to DIB): Click **Send**.
    The installation successful page appears.



6

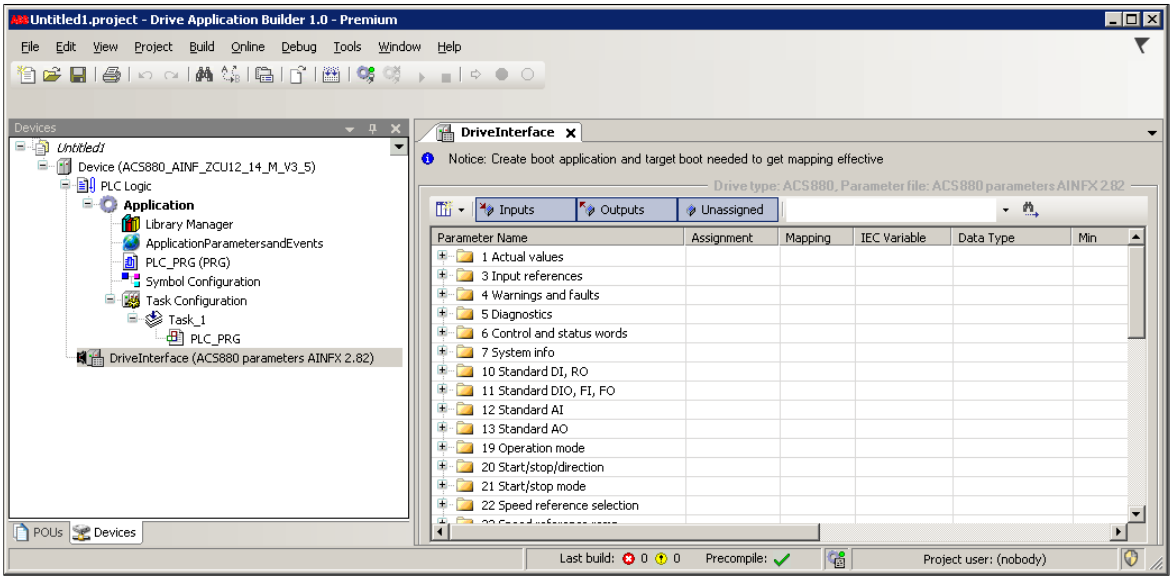
DriveInterface

Contents of this chapter

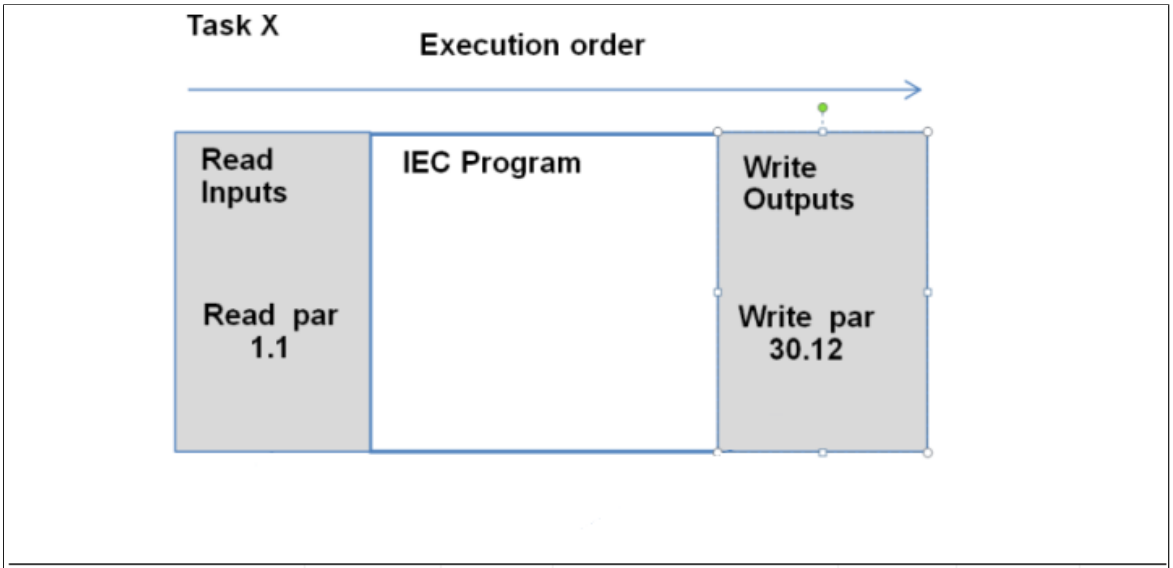
This chapter describes how to implement DriveInterface and map input/output settings between the application programs and the drive firmware parameters.

Implementing DriveInterface

The interface between the drive firmware and the application is implemented using DriveInterface.



DriveInterface consists of all the drive firmware parameters list that can be used in the application program. The list is specific to each drive firmware (a new firmware may have new parameters). You can assign a parameter to be an input for the application program and define that the parameter is read at the beginning of the task execution. Similarly, you can assign parameters to be an output of the application.



Parameter Name	Assignment	Mapping	IEC Variable	Data Type	Min	Max
1 Actual values						
1 Motor speed used	Input		Device_Motor_speed_used_1_1	REAL	-30000.00	30000.00
30 Limits						
12 Maximum speed	Output		Device_Maximum_speed_30_12	REAL	-30000.00	30000.00

Note: The parameter to IEC variable mappings is valid only after creating a boot application. For more details, see section *Application download options (page 53)*.

- Drive interface is not completely covering all the drive parameters. If the firmware parameter is not available in the drive interface list, you can use AY1LB library functions to read/write firmware parameters.
- In order to fully remove drive parameter settings from the drive, use Reset origin option. Also, re-save user sets (see parameter 96.08) after removing or replacing the application. As user set may have incorrect mapping of firmware parameter to nonexisting application.

Selecting the parameter set

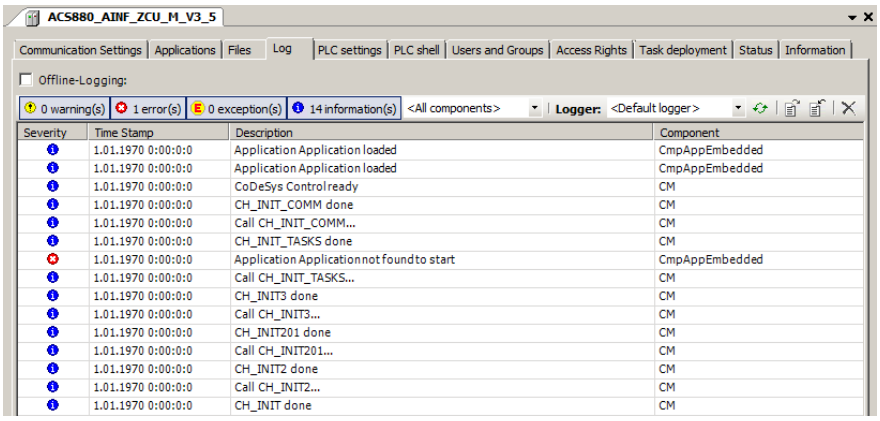
A drive can have different parameters depending on the firmware version. Before performing parameter modification, make sure that the correct parameter set is selected in DriveInterface. The changes to parameter set in DriveInterface removes all the parameter mapping data.

To change the currently selected parameter set, follow these steps:

1. In the Devices tree, right-click DriveInterface and select **Update objects**.
   Update object window is displayed.
2. Select the correct parameter set for the current target and click **Update objects**.

Viewing parameter mapping report

After downloading the application program, a report of unresolved parameter mappings between project parameters and actual parameters, messages, errors and warnings in the drive are written in the Log.

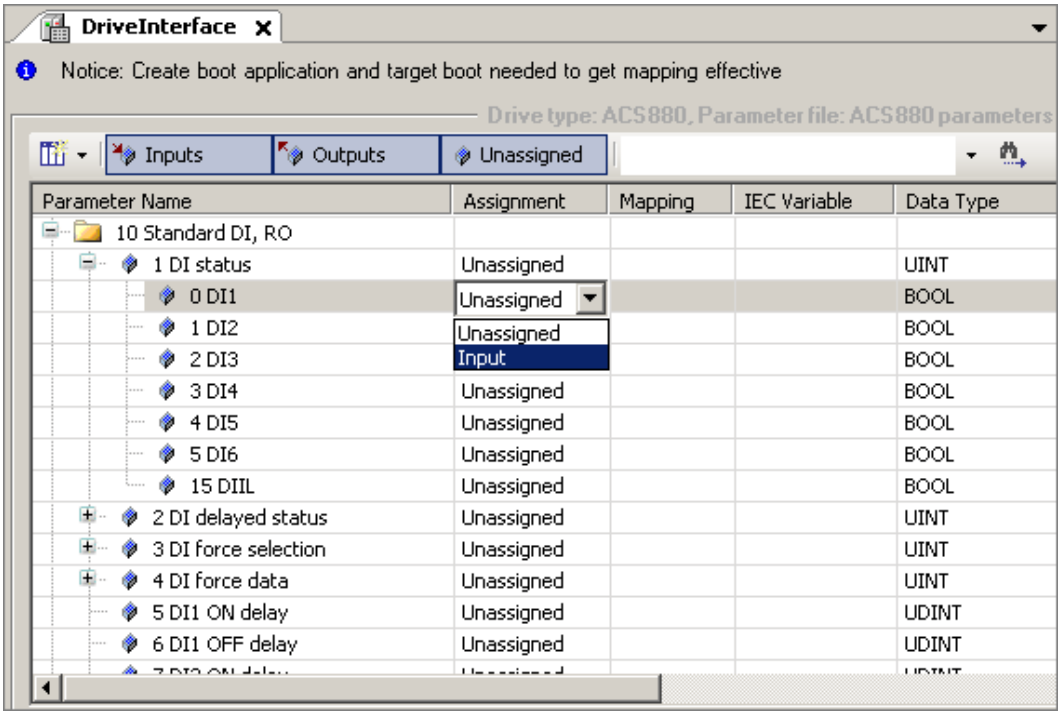


For more details on downloading application program, see sections *Downloading the program to the drive (page 40)* and *Application download options (page 53)*.

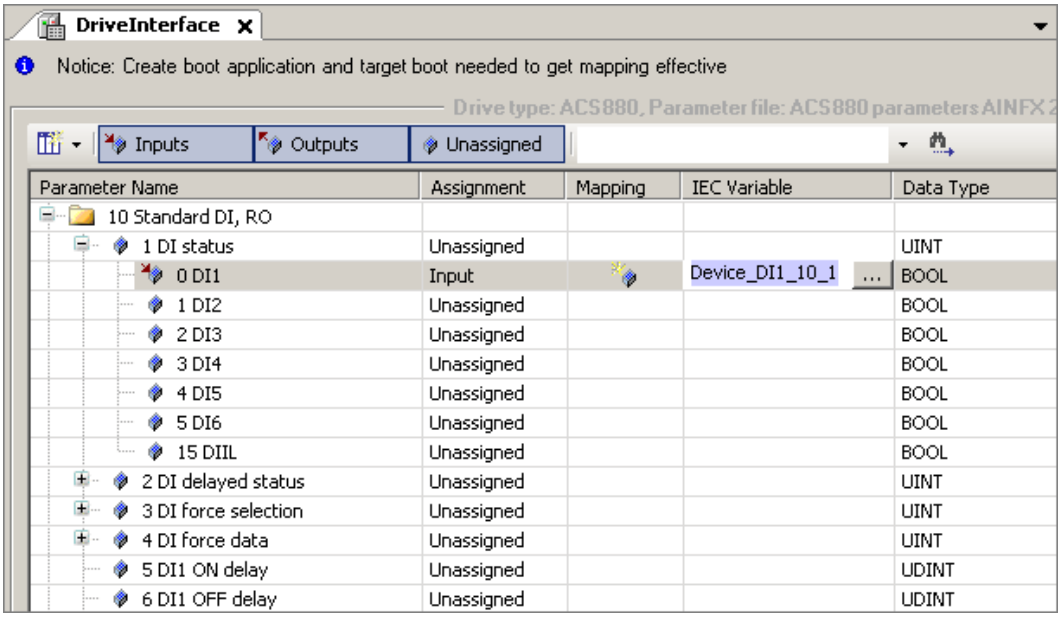
Mapping example

To read digital input DI1 of the ACS880 control unit to the previous CFC program example (see chapter *Creating application program (page 25)*), open group 10 and select index 1.

1. In the Devices tree, double-click **DriveInterface**.
2. In the Driveinterface window, double-click on the required Assignment cell and select Input or select the desired Assignment from the available drop-down list.



3. Double-click default IEC variable name (eg, Device_DI1_10_1).
   A button is displayed to change the name.



4. Click input assistant button to change the name. Input Assistant dialog is displayed.
5. Click **Categories** and expand DriveInterface tree to select a Device and then click **OK**. IEC variable name is changed.

If you want to select existing variable DI1 from the POU variable list, expand Application and under POU, select DI1. DI1 is connected to drive parameter *10.1 DI status* bit 0.

The mapped parameters are available as IEC variables in the program editors (press F2).

Note: Bit and value pointer parameters can be used as outputs and then the pointer is linked directly to the application memory.

Updating drive parameters from installed device

You can update the parameter list from the installed device or you can take the actual drive parameter set used in DriveInterface from Drive composer pro. See section *Updating drive parameters from parameters file*.

To update the parameters from the installed device, follow these steps:

1. In the Devices tree, right click DriveInterface and select **Update Drive Parameter Set**. Update parameter set dialog is displayed. By default **From installed device** option is activated.
2. Expand Miscellaneous and select the device.
3. Click **Update**.
   The parameter list from the selected device is displayed.

Updating drive parameters from parameters file

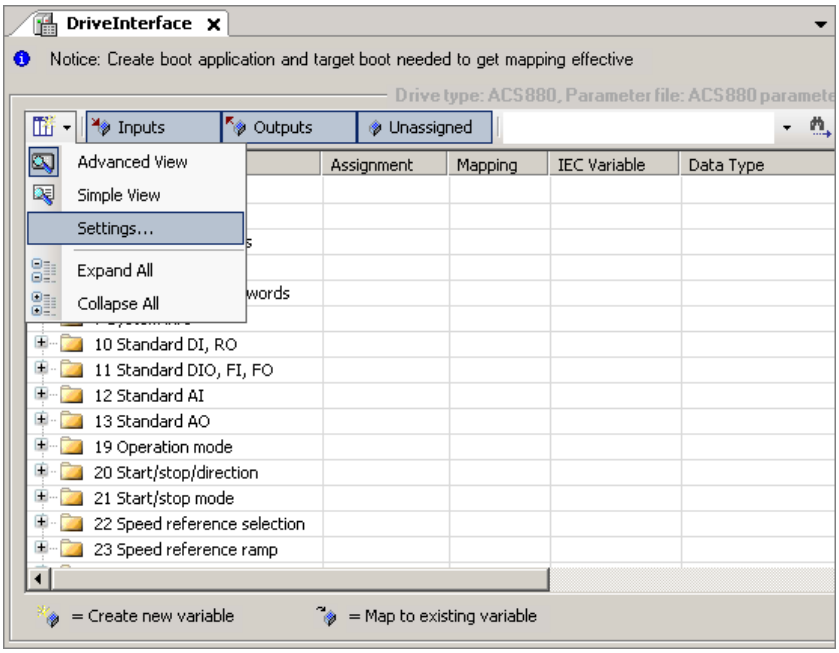
Optionally, you can update the actual drive parameter set using the Drive composer pro backup file.

To update the parameters backup file, follow these steps:

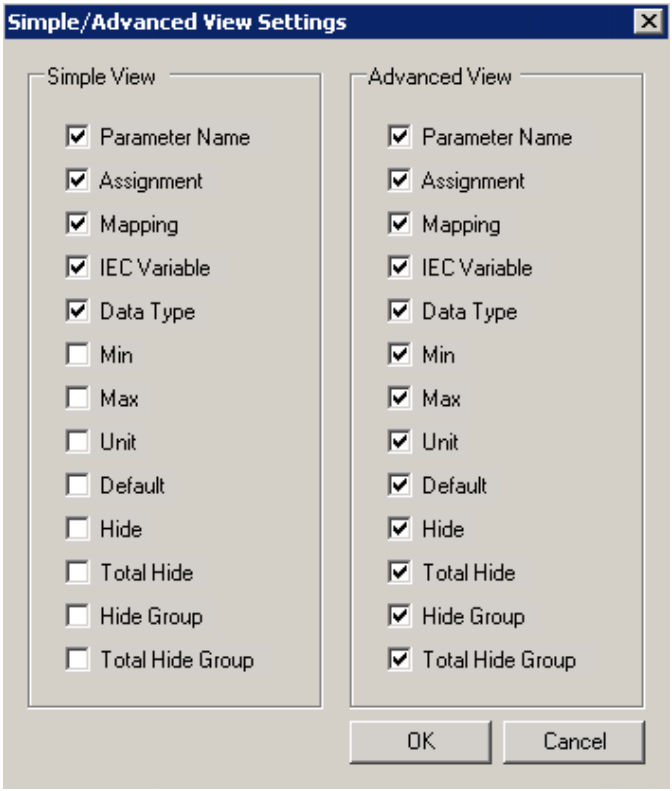
1. In the Devices tree, right click DriveInterface and select **Update Drive Parameter Set**. Update parameter set window is displayed.
2. Select From parameter file option and browse to select dcparams (*.xml*) or Drive composer backup file.
3. Click **Update**.
   The changes/deleted parameters are displayed. Click **OK**.

Setting parameter view

1. In the Devices tree, double-click **DriveInterface**.
2. In the upper-left corner of the DriveInterface window, select **Settings**.



3. Select the required view option for the corresponding parameter and click **OK**.



The selected options in the view list are displayed in the DriveInterface parameter window.

7

Application parameters and events

Contents of this chapter

This chapter describes how to use parameter manager and provides detailed information on parameter settings.

Application parameters and events

You can create application parameters and events visible in the panel and Drive Composer pro tools.

1. In the Devices tree, right-click Applications and click **Add Object**.
2. Select Application Parameters and click **Add object**.
   Add Application Parameters dialog is displayed. Click **Add** to add the Application Parameters to Devices tree.
   Note: You can create only one ApplicationParametersandEvents object at a time.

Parameter manager

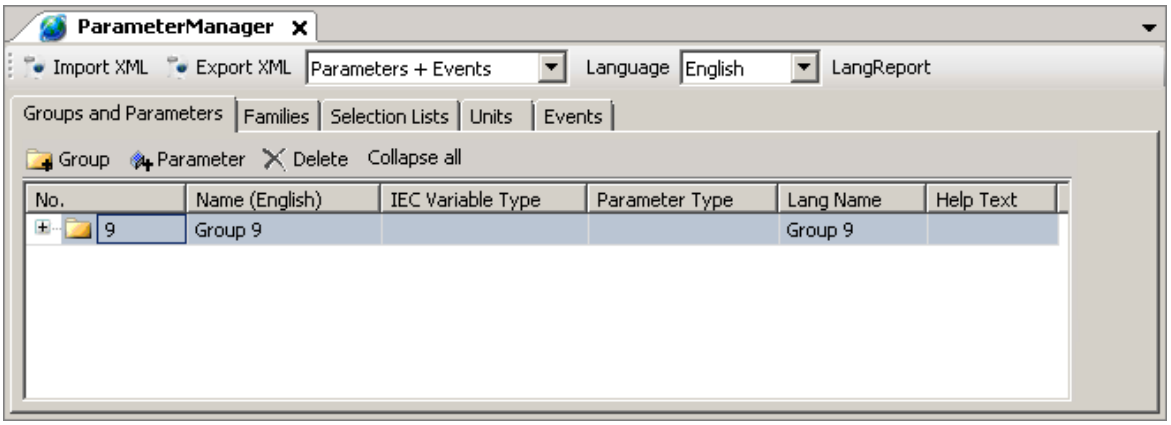
In the ParameterManager window, you can create new groups with parameters, parameter families, selection lists, units, events and language translations for the names of all the previous items.

In the Devices tree/Application, double-click **ApplicationParametersandEvents** object. The ParameterManager window is displayed.

Creating parameter groups

All the drive parameters belong to a specific parameter group. Before creating new parameters, create a new parameter group. Make sure that all the groups have unique name and number. You can change the group number and name. You can also add translations into other languages in addition to the default language which is English.

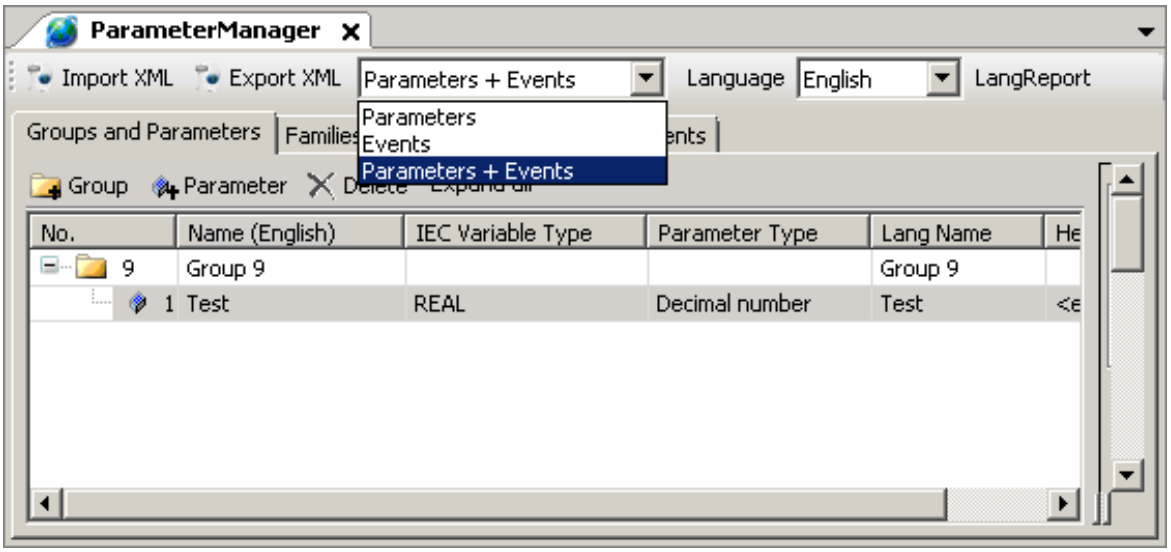
In the ParameterManager window, click **Group** button to add a group.



ParameterManager automatically selects the first free parameter group number that is not used in the drive firmware or ParameterManager.

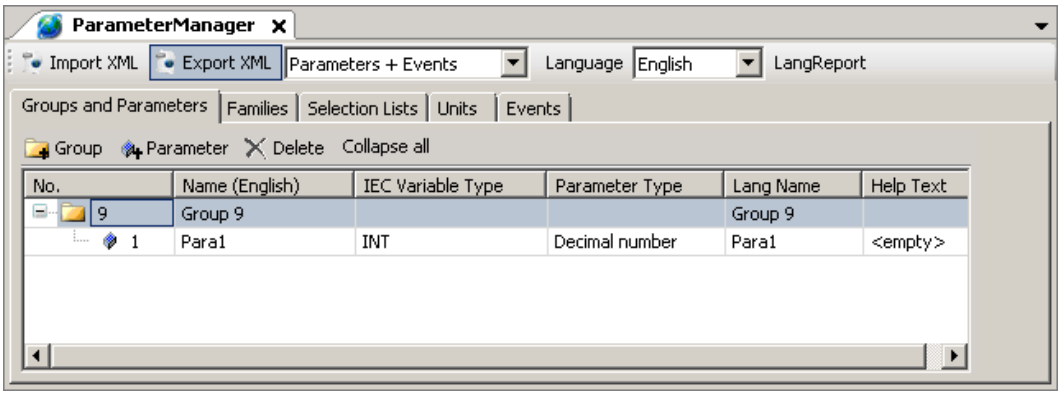
Importing and exporting parameters

You can import and export Parameters, Events and Parameters + Events in the form of XML format. Choose the desired option from the drop-down list and click **Import XML** or **Export XML**.



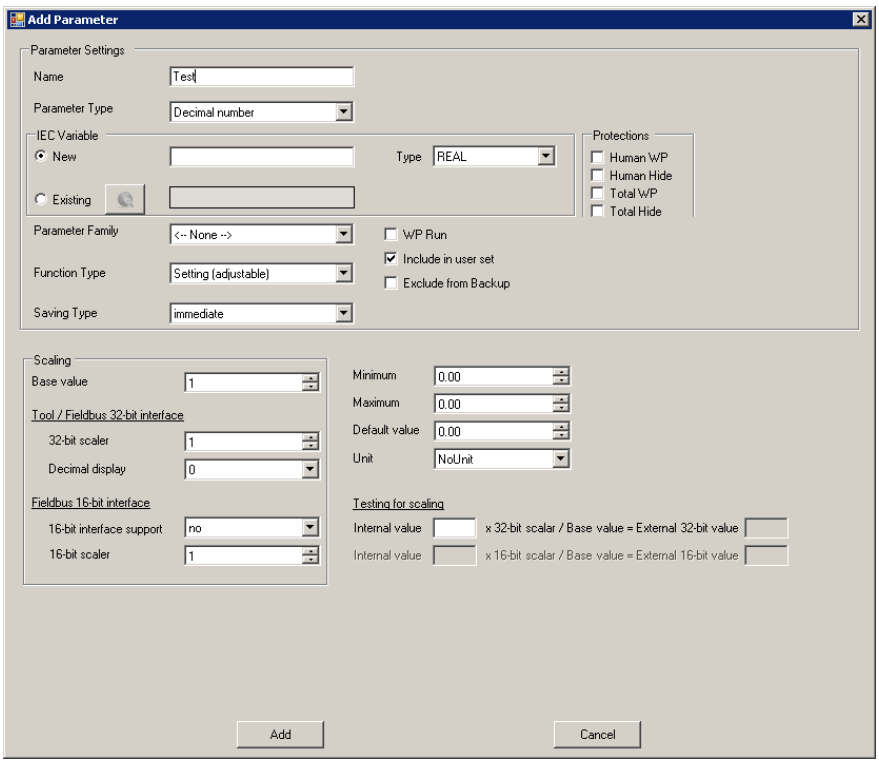
Creating parameters

1. In the ParameterManager window, select a parameter group.
2. Click **Parameter** button to create a new parameter.
   Parameter Settings window is displayed.
   In the Parameter Settings window, you can set the properties of the parameter. The Parameter Settings window is identical for all the parameters but there are also custom settings available depending on the parameter type.
   For more information on parameter settings see section *Parameter settings* and for the parameter type-specific windows, see section *Parameter types*.
3. In the Parameter Settings window, enter the Name of a parameter and click **Add**.
   A new parameter is added to the selected group.



Parameter settings

In the Parameter Settings, you can set parameter properties.



Parameter name

The name shown in the parameter list when using Drive composer pro or ACS-AP-x control panel.

Parameter type

Defines the type of parameter.

The following parameter types are available in the drop-down list.

- Decimal number
- Formatted number
- Bit pointer
- Value pointer
- Plain value list
- Bit list (16 bit)

For more information, see section *Parameter types*.

IEC variable name

Used to define IEC variable for the parameter.

- The New option maps the parameter to a new IEC variable. If you do not give a name for the new IEC variable, the parameter name is used as the IEC variable name.
- When you create a new IEC variable, you must select the variable type, for example, REAL. The selected parameter type restricts the variable type selection and only the allowed types are shown in the IEC variable/Type list. For more information on the variable types, see chapter *Features (page 47)*.
- The Existing option maps the parameter to an already existing IEC variable by finding the parameter from the list of the Input Assistant or writing the name to the field.

Parameter family

Includes a parameter as part of the parameter family and inherits the settings defined for the family. For more information, see section *Parameter families*.

Function types

These are flag configurations for parameters which determine the parameter behavior with the ACS-AP-x control panel and PC tool displays. There are five different configurations:

- Setting (adjustable) - This function type is a generic configuration parameter. When a parameter with this function type is changed by ACS-AP-x control panel or Drive composer, the changed value is saved.
- Setting (reverts to default) - Used to request a function. When this request is processed, the parameter returns to its default value.
- Signal (read only) - Displays the application parameter value in the ACS-AP-x control panel or Drive composer pro. A parameter of this function type does not have any meaningful default value.
- Signal (resettable) - This function type is identical to the read-only signal and allows to reset parameters to their default values.
- Custom - Enables to change the values in the application.

Saving types

Define the method of storing the parameter value to the non-volatile memory.

There are three different saving types:

- No - Does not store the parameter value changes done in the ACS-AP-x control panel or Drive composer pro.
- Powerfail - If the parameter *95.04* is set to Internal 24V, the powerfail type parameters are saved immediately at the time of power failure in the drive. If the parameter *95.04* = External 24V, the values are saved at periodic intervals of 1 minute. The powerfail saved parameters are limited to < 10.
- Immediate - If the parameter value is changed using keypad or PC tool, this type saves the value immediately within 10 seconds. This saving type is used for controls, but not for signals.

Protection, hiding and excluding from backup

Allows you to set the following protections for parameters or set them on the parameter group level by selecting a parameter group in ParameterManager.

- Human WP/Human Hide write protects/hides the parameter from a human user manipulation. This setting can be bypassed using configuration tools, fieldbus controllers, and so on.
- Total WP/Total Hide write protects/hides the parameter from any kind of manipulation outside the firmware. These parameters are used only by the application.

The following settings are for parameters only:

- WP Run protects the parameter from writing when the drive is running.
- Include in user set includes parameter as a part of the process where all the parameters become a user set.
- Exclude from Backup leaves the parameter out of parameter backup, but restores the default parameter values. This setting applies only for parameters.

Minimum, Maximum and Default value

These are set for decimal and formatted numbers.

- Minimum and Maximum define the limits for the value of the parameter. These values should not exceed the limits of the data type defined for the parameter.
- Default value is the value of the parameter at the start-up of the program and it must be within the limits defined by the minimum and maximum values. The default value returns if you restore defaults or clear all with parameter *96.06* (see the drive firmware manual).

Scaling

Base value is the internal firmware value. The scaling values in Base value, 32-bit scaler and 16-bit scaler must match each other and define how a value of the parameter is represented in other contexts. Scaling the other values of a parameter is calculated based on the defined scaling values.

If the scaling factor is 1, meaning direct transform from one representation to another, use the same number for all of the scaling values

Example:

The firmware uses values 0…1 for motor rotation speed measurement. The maximum speed is 1500 rpm, and therefore the ACS-AP-x control panel displays 1500 rpm when the internal value is 1 (the maximum speed). The 16-bit fieldbus device shows 100%.

In this example the values are: Base value = 1, Value (32-bit int) = 1500, Value (16-bit int) = 100

Tool/Fieldbus 32-bit interface

- 32-bit scaler - 32-bit external value (for example, Drive composer pro or ACS-AP-x control panel)
- Decimal display - Defines the number of decimal digits displayed on the Drive composer pro or ACS-AP-x control panel. This setting applies only for an external value, but has no effect on the internal value.

Fieldbus 16-bit interface

- 16-bit interface support - This field defines if the 16-bit external format is allowed, for example, in fieldbus devices and how it is scaled to the 32-bit external format:
  - No - 16-bit external format is not allowed.
  - Direct - 32-bit scaling is used but the value is displayed as a 16-bit value. Therefore, the value (16-bit int) is considered meaningless.
  - Scaled - separate 16-bit scaling is used. Value (16-bit int) must be defined.
- 16-bit scaler - 16-bit external value (for example, fieldbus devices).

Testing for scaling

Internal value calculates the scaling of 32 and 16 bit fieldbus interface with the corresponding IEC variable. For description of formula, see *PAR_SCALE_CHG* function block.

Linking parameter to application code

The IEC variable field in the Parameter settings window enables to link a parameter to an application program code. There are two options to link a parameter with an application program code.

- The New option adds a new IEC variable to the program and is visible in the input assistant under ApplicationParametersandEvent object.
- The Existing option allows linking a parameter to the existing IEC program variable using browser. Make sure to select the correct data type. If you change the link to the existing IEC variable, a build error occurs. For information on incorrect linked parameters, see the message box. Check the full path to correct the missing linked parameters according to the program.

  Note: The existing retain variables cannot be linked to application parameters.

Parameter types

In the Parameter Settings window, you can select the Parameter Type for the newly created parameter.

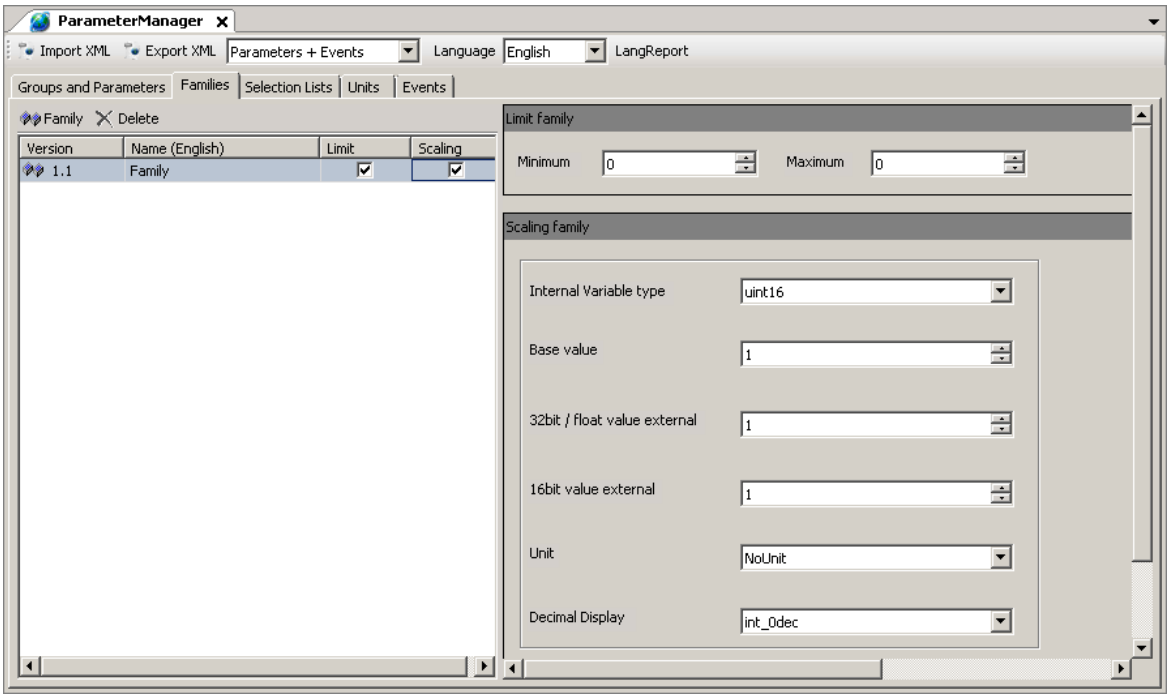
- **Decimal number** creates a parameter with actual numeric contents, either decimal or non-decimal numbers. The available IEC types are REAL, UDINT, UINT, DINT and INT.
- **Formatted number** parameter type is used to make special purpose parameters like date displays, version texts, passcodes, and so on. The available IEC types are UDINT, UINT, DINT and INT. In the Display format for Data Parameter, you can define the format in which the value should be displayed in the Drive composer or ACS-AP-x control panel.
- **Bit pointer** creates a pointer parameter which can be assigned to point to a bit of another parameter. You must associate the bit pointer parameter to a selection list (a bit pointer list) that must be created beforehand. For more information, see section *Selection lists*. The only available IEC type for bit pointer is BOOL. You can define the default selection from the list.
- **Value pointer** creates a pointer parameter which can be assigned to point to another parameter. You must associate the value pointer parameter to a selection list (a value pointer list). For more information, see section *Selection lists*. The only available IEC type for the value pointer is UDINT. You can define the default selection from the list.
- **Plain value list** must be associated to a selection list (a plain value list). It allows only values of a list as its own value. The available IEC types are UDINT, UINT, DINT and INT. You can define the default selection from the list.
- **Bit list (16 bit)** consists of maximum 16 Boolean values (bits). You can add new rows (bits) to the list using the Bitlist row button. You can change the names of the bits and their values to represent their purpose. The default value is the bit value at the start-up of the program. The only available IEC type is UINT.

Parameter families

If a parameter shares some of its attributes (scaling, minimum/maximum, and so on) with another parameter, it can belong to a family that describes these common attributes. This way, when the attribute is changed in one parameter, it is also changed in all parameters belonging to the same family.

The system library includes a function block to modify parameter attributes like PAR_UNIT_SEL functions. See AY1LB_System_ACS880_V3_5 library in *Appendix B: ABB drives system library*.

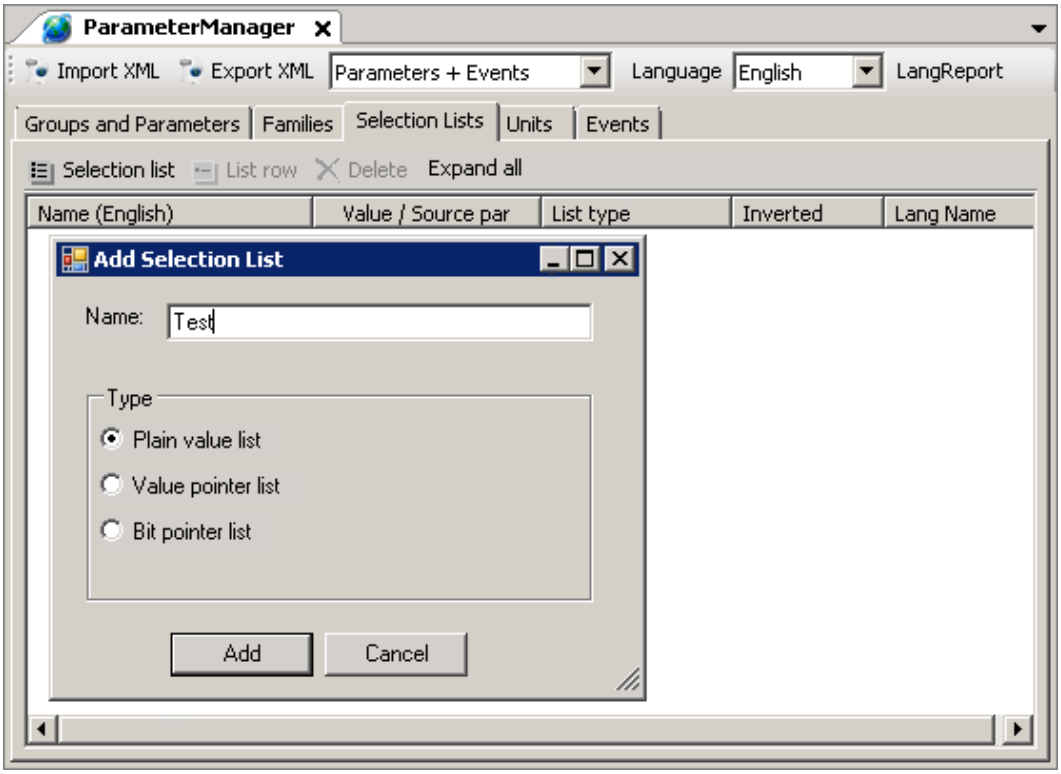
If you select a parameter family Version style, make sure the family has a unique Name. The parameter families can define limit or scaling properties or both.



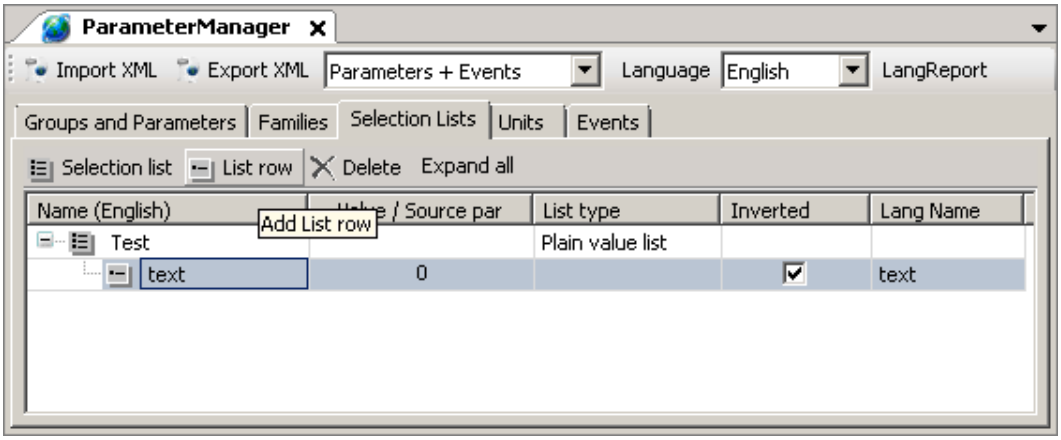
Selection lists

Selection lists are always associated to a parameter of the same type as in the list. They are accessed only through the parameters.

1. In ParameterManager window, click **Selection Lists** tab and then click **Selection list** to add values.
2. Select the Type of selection list and enter the name and then click **Add**.



The selection list is created. You can add the list row by clicking on List row button. If you want to rename the list, double-click on the created list.



Note: You cannot change the type of selection. If you want to change the type of selection, delete and create a new selection list.

- Name (English) - The text visible to the user. Note that the name is not the official text since the language translator use this text as a source when creating the official language texts.

- Value/Source par - The value of the list row. For the bit and value pointers, it is the index of the row in the list. For the value lists, it is an actual selectable value.
- List type - There are three different types of selection lists:
  - Bit pointer list - By default, the bit pointer list has the const_false and const_true values. You can add single bits of any parameter of the appropriate type.
  - Value pointer list - By default, the value pointer list has the const_null value. You can add any parameter which has the same data type as the pointer associated to the list.
  - Plain value list - You can add any values of type INT, DINT, UINT or UDINT. The type should be same as the type of the pointer associated to the list.

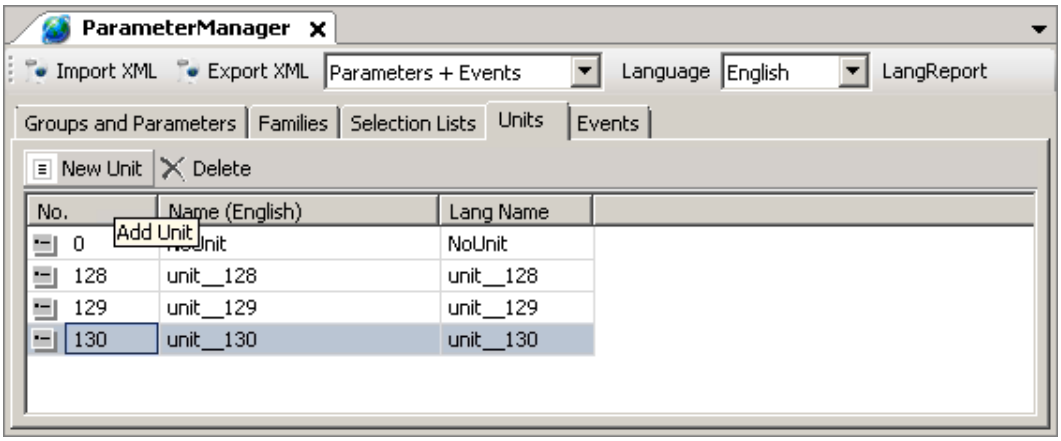
3. Inverted - When a bit /value is read from a source parameter, it is inverted /negated for output when the inverted flag is set.

Units

You can create own units for the application parameters. A unit has a unique number and a name. The allowed unit codes for the application program are 128…255.

You can add translations of the name into other languages.

1. In the ParameterManager view, click **Units** tab.
2. Click **New Unit** to add unit and click **Add** to add Language Id.

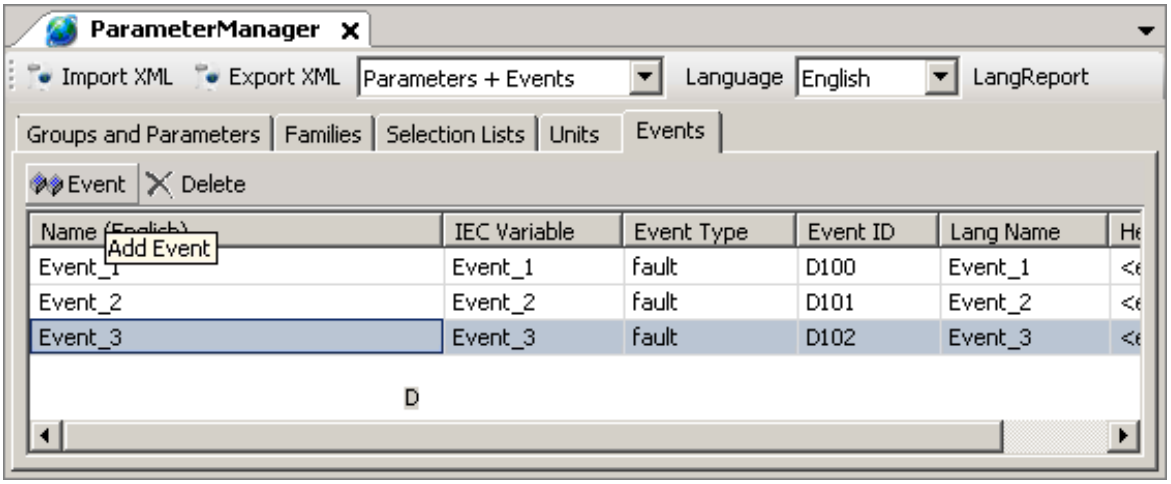


The units are attached to parameters in the Add Parameter options in Parameter Settings window.

Application events

You can configure your own application events (faults or warnings). The application program then triggers the event and the event registers in the drive event logger with a time stamp. The tool defines the event ID code, type and event name (with translation).

In the ParameterManager view, click **Events** tab and then click **Event** to add Event.



Events dialog box gives the following information:

- Name, in this example Event_1. The Event name is displayed on the ACS-AP-x control panel and in the Drive composer tools when the event is activated/deactivated.
- Event Type, in this example fault.
  The following event types are supported:
  - 1 = Fault (Trips the drive)
  - 2 = Warning (Is registered to the event logger)
  - 3 = Pure event (Is registered to another logger)
- Event ID, in this example D100. Each type of event has its numerical range (ID code). You can select the ID code within the range.

The event is activated by using the EVENT function block in the program code (library AY1LB_System_ACS880_V3_5, see chapter *Libraries (page 107)*). Every event must have its own instance of the EVENT block. The EVENT function block must have the same ID code and type as defined in the previous dialog box.

8

Configuring extension I/O modules

Contents of this chapter

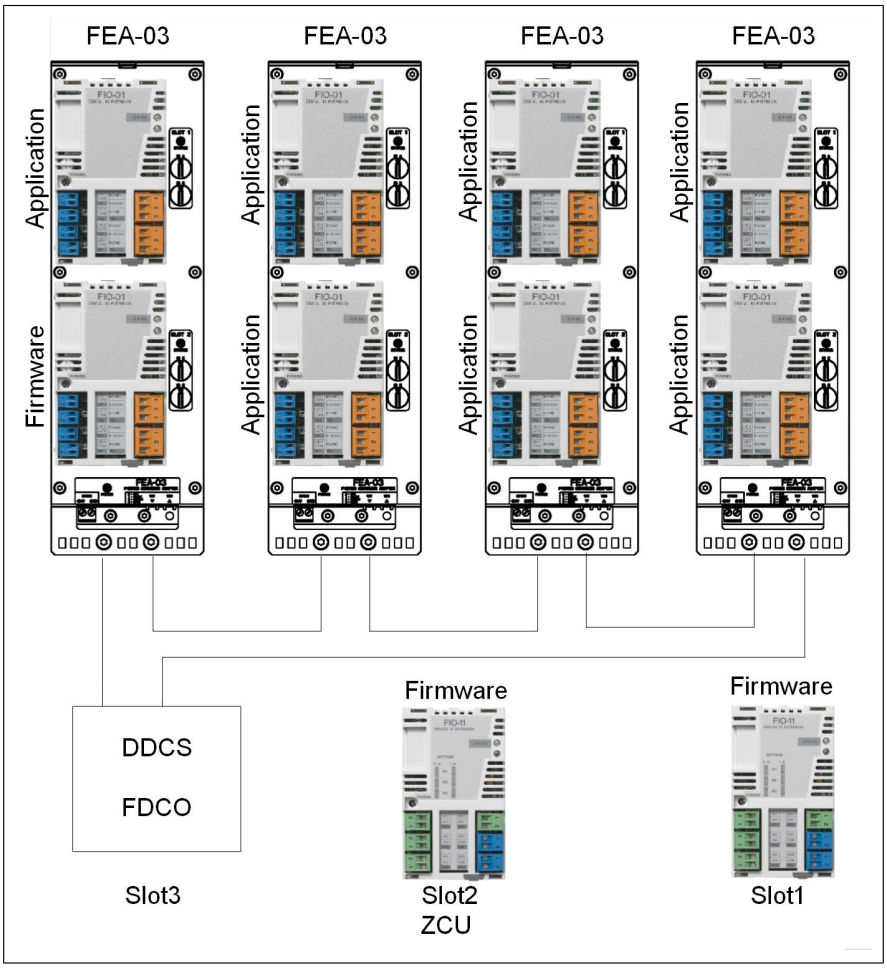
This chapter contains general information on how to configure F-Series extension I/O in drive application programming through Drive Application Builder programming tool.

Configuring extension I/O module

FEA-03

The FEA-03 F-series Extension adapter is used to locate additional F-series modules like FIO-01, FIO-11 or FAIO-01. The FEA-03 module contain 2-slots with 2-switches each. You can add FIO-01, FIO-11 or FAIO-01 modules to the slots of the control board or FEA-03 module. The application programming supports 7-extension I/O modules. See parameter group *14 I/O extension module 1* in *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].

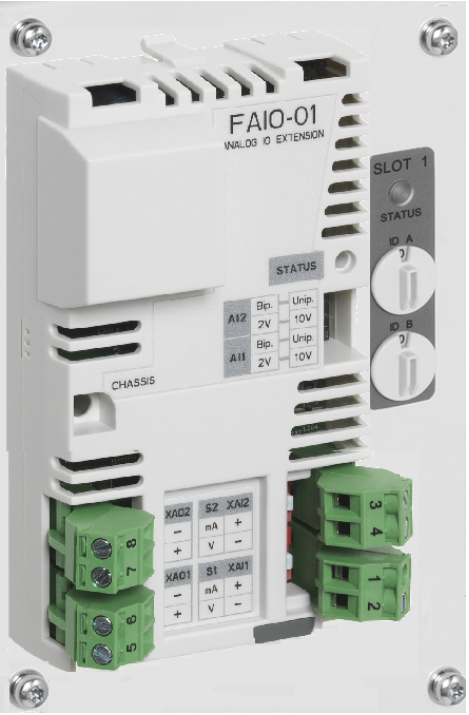
For example, the figure below illustrates the maximum configuration of F-Series modules on the Control board (ZCU) and FEA-03 adapters. It contains 3-firmware and 7-program modules. Node numbers 1, 2, 3 are on control board slot 1, 2, 3 and the remaining node numbers are FEA-modules and their node numbers are defined by F-Series module switch.


FEA-03
FEA-03
FEA-03
FEA-03
Application
Application
Application
Application
Firmware
Application
Application
Application
DDCS
FDCO
Firmware
Firmware
Slot3
Slot2
ZCU
Slot1

Node numbers

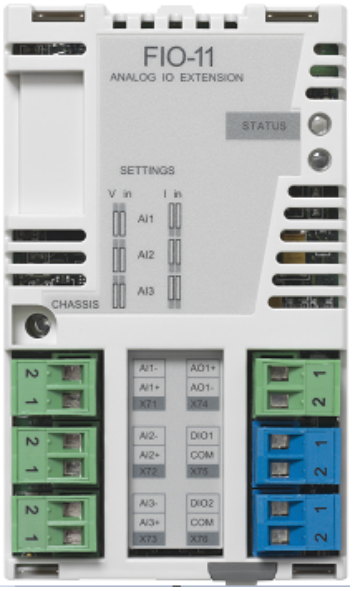
The node numbers 1…3 are reserved for extension I/O modules that are placed on the slots of control board and the other node numbers can be used for modules in FEA object.

The upper switch defines the first digit and the lower switch defines the second digit of the node ID. For example, in case of node address 6, turn the lower switch to 6 and check that the upper switch points to 0.

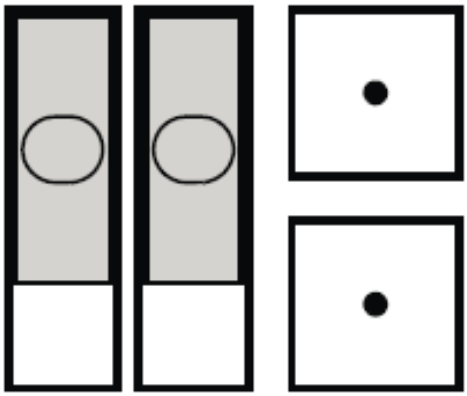


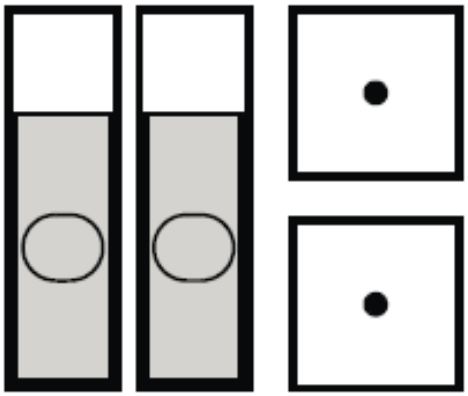
Selecting input signal type

You can select the unit (mA or V) of an analog signal by sliding the switches of FIO module next to the input either up for current signal or down for voltage signal.



Current and voltage signal switches.

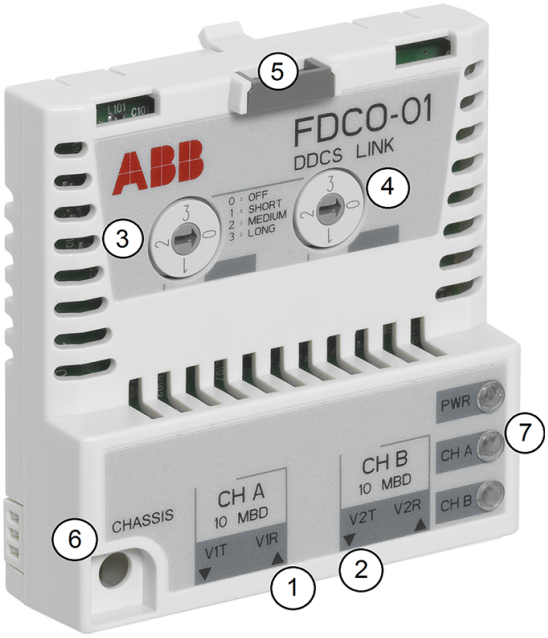




FDCO

In FDCO adapter, select the channel number based on the used slot. Communication slot for FDCO adapter is defined by parameter *60.41 Extension adapter com port* based on the used slot and channel. For the descriptions of parameter, see *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].

For example, if FDCO adapter is placed on slot 2 and channel A is used, then slot2A is selected for Extension adapter com port. For further details, see *FDCO-01/02 DDCS communication modules user's manual* [3AUA0000114058 (English)].

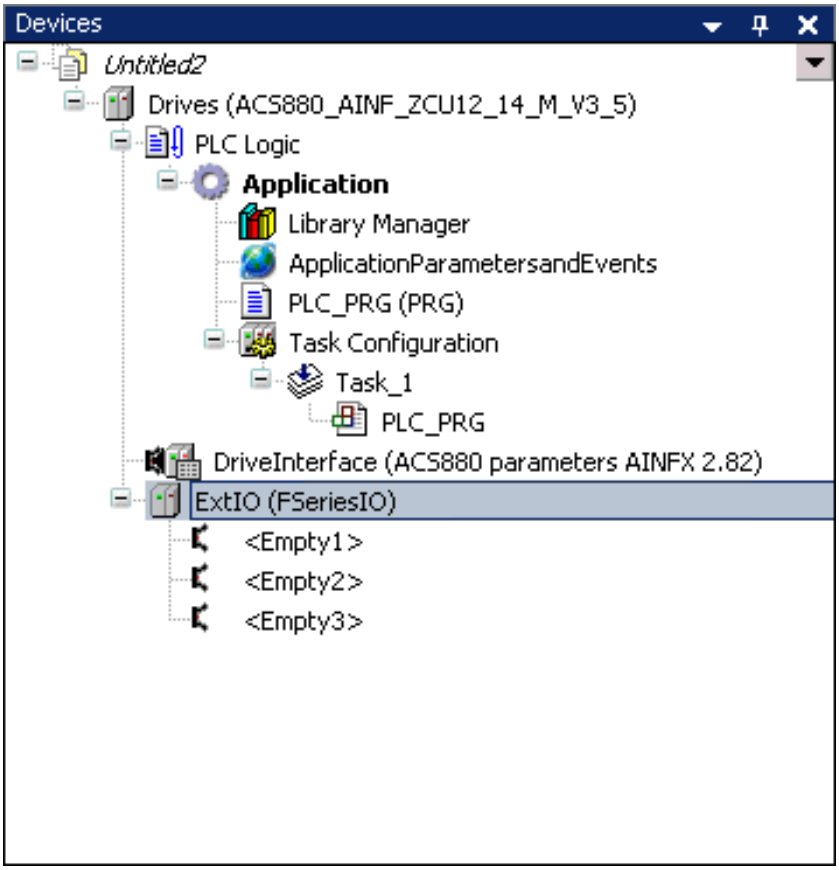


1	Connector for channel A
2	Connector for channel B
3	Selector for channel A
4	Selector for channel B
5	Lock
6	Mounting screw
7	LEDs

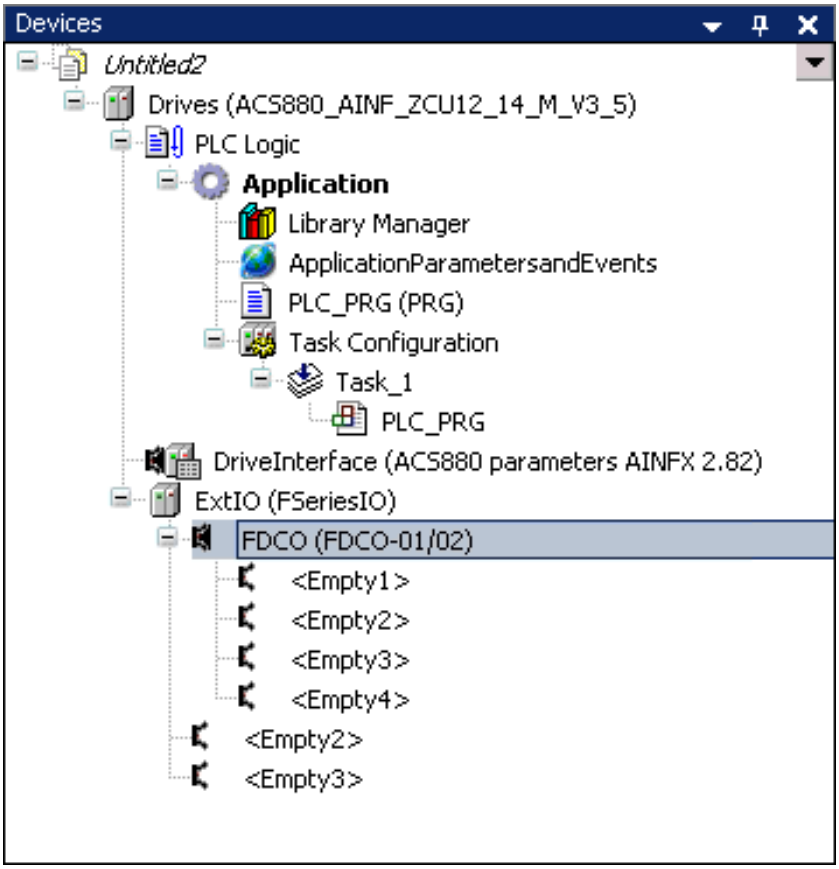
Extension I/O in drive application program

Adding F-series module

1. In the Drive Application Builder Device tree, right-click on device node and select **Add object**.
2. Select FSeriesIO and click **Add object**.
   The FSeriesIO extension is added to the project. It contains 3-empty slots. You can add FIO-01, FIO-11 or FAIO-01 modules to F-Series slots. FDCO adapter is required if you are using FEA-03 module.



   Note: You can add only one FDCO adapter to FSeriesIO extension. Because it has only one communication port for FDCO adapter in the firmware. See parameter *60.41 Extension adapter com port* in *ACS880 primary control program firmware manual* [3AUA0000085967 (English)].
3. In the ExtIO (FSeriesIO), right-click on empty slot and click **Add object**.
4. Select FDCO-01/02 adapter and click **Replace object**.
   FDCO-01/02 adapter is added to the Slot of FSeriesIO module.



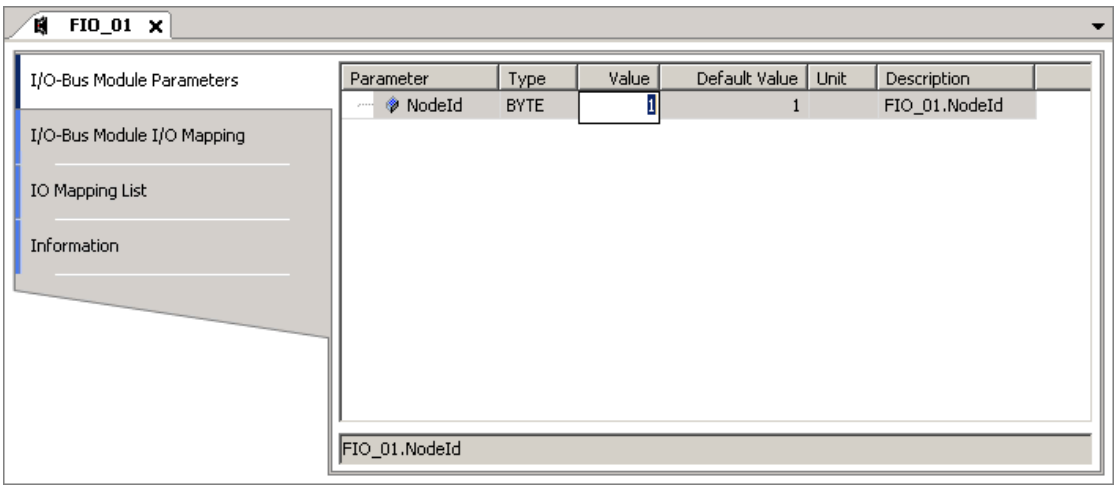
5. In the FDCO (FDCO-01/02), right-click on empty slot and click **Add object**.
6. Select FEA-03 and click **Replace object**.
7. In the FEA (FEA-03) module, right-click on an empty slot and click **Add object**.
8. Select FIO-01 module and click **Replace object**.
   Similarly, you can add FIO-11 or FAIO-01 modules to FEA-03 empty slots.

Note: F-series IO modules used in Drive Application Builder should not be activated as extension IO module in groups 14, 15 and 16 in Drive composer pro.

Setting module data

Adding node number

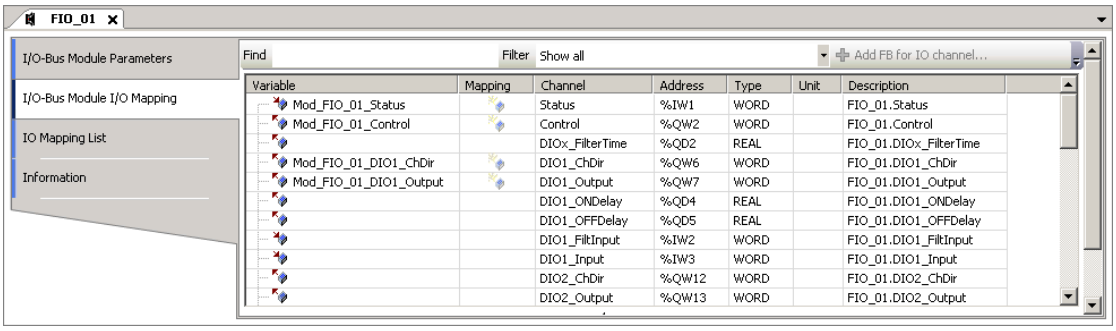
1. In the Drive Application Builder, double-click FIO_01 or any other module.
2. Click **I/O-Bus Module Parameters** tab and add the node number in the value field.



The node numbers 1, 2 or 3 are based on slot numbers. The node numbers 4…10 are used if the I/O module is placed on FEA-03 module.

I/O mapping variables

1. In the Drive Application Builder, double-click **FIO-01** or any other module.
2. Click **I/O-Bus Module I/O Mapping** tab and create I/O mapping variables in Variable column.



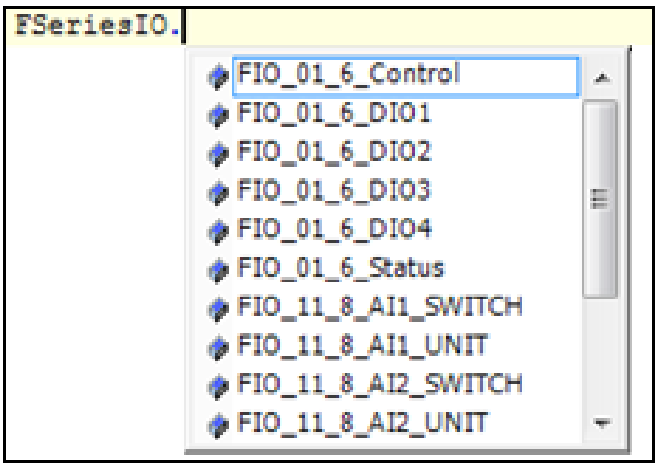
The variable names must be individual. You can have maximum 100 mapping variables. The I/O mapping variables do not support Mapping to existing variables.

Using F-series I/O from the application

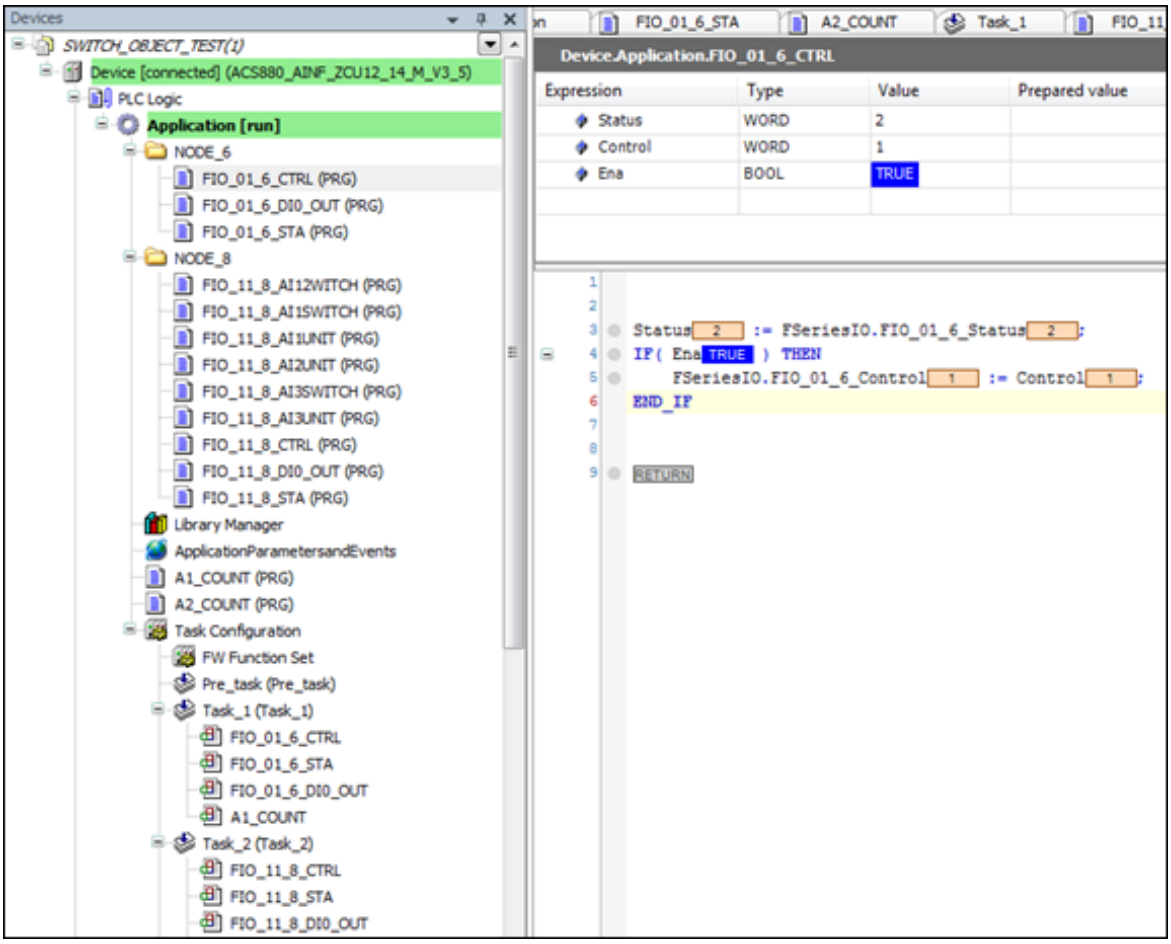
You can assign I/O module related blocks into the same application task. Do not assign F-Series related block into PreTask. The I/O module can be handled according to the fastest task cycle which contains module related blocks.

For example, FIO-01 is using Task_1 cycle and FIO-11 is using Task_2 cycle. If some of the FIO-11 handling blocks are moved into Task_1, then FIO-11 is handled (internally) using Task_1 cycle, regardless, where other FIO-11 blocks are assigned. (Task_1 has fastest cycle).

The IEC variable name must start with F-Series I/O.



The below figure shows the status of the application used for controlling the application execution or producing application based events.



Adding bus fault control

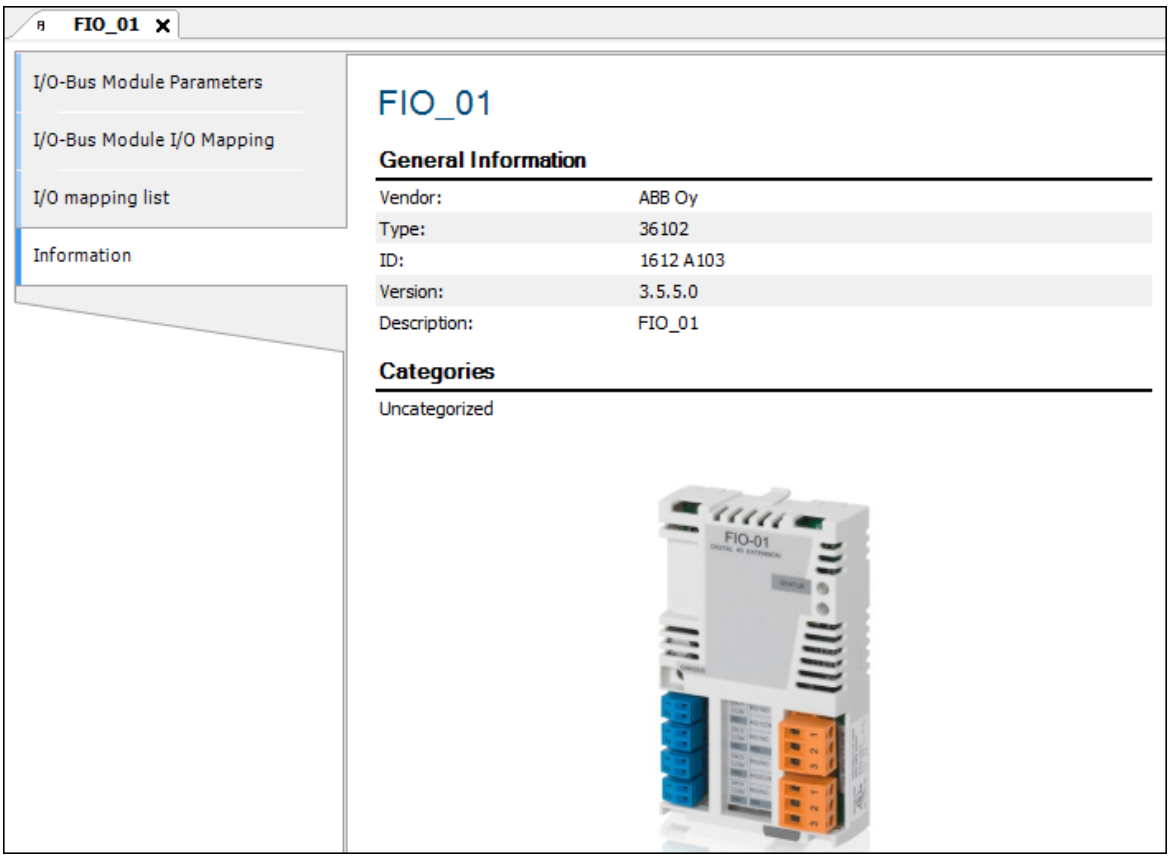
To add F-Series I/O module to Drive Application Builder project, proceed as follows:

1. In the Drive Application Builder Devices tree, double-click **ExtIO (FSeriesIO)**.
2. Go to I/O-Bus Module I/O Mapping tab and create I/O mapping variables in Variable column.
   FSeriesIO.BUS_Control value should be 0 in a program. The program should store value 1 in FSeriesIO.BUS_Control if the FSeriesIO.BUS_Status value is 3 (no communication). The program should restore value 0 if the FSeriesIO.BUS_Control after the bus break is over when SeriesIO.BUS_Status I/O receives the value 2 (active).
   The FSeriesIO.BUS_Status receives the same values as channel status.
   - 0 = Not active
   - 1 = Initializing state
   - 2 = Active
   - 3 = No communication

Note: The F-Series I/O bus does not recover automatically after the bus break. The bus can recover without motor stop and restart by using Bus Control variable.

FIO-01 Module data

You can find the general information of FIO-01 module by clicking on **Information** tab.



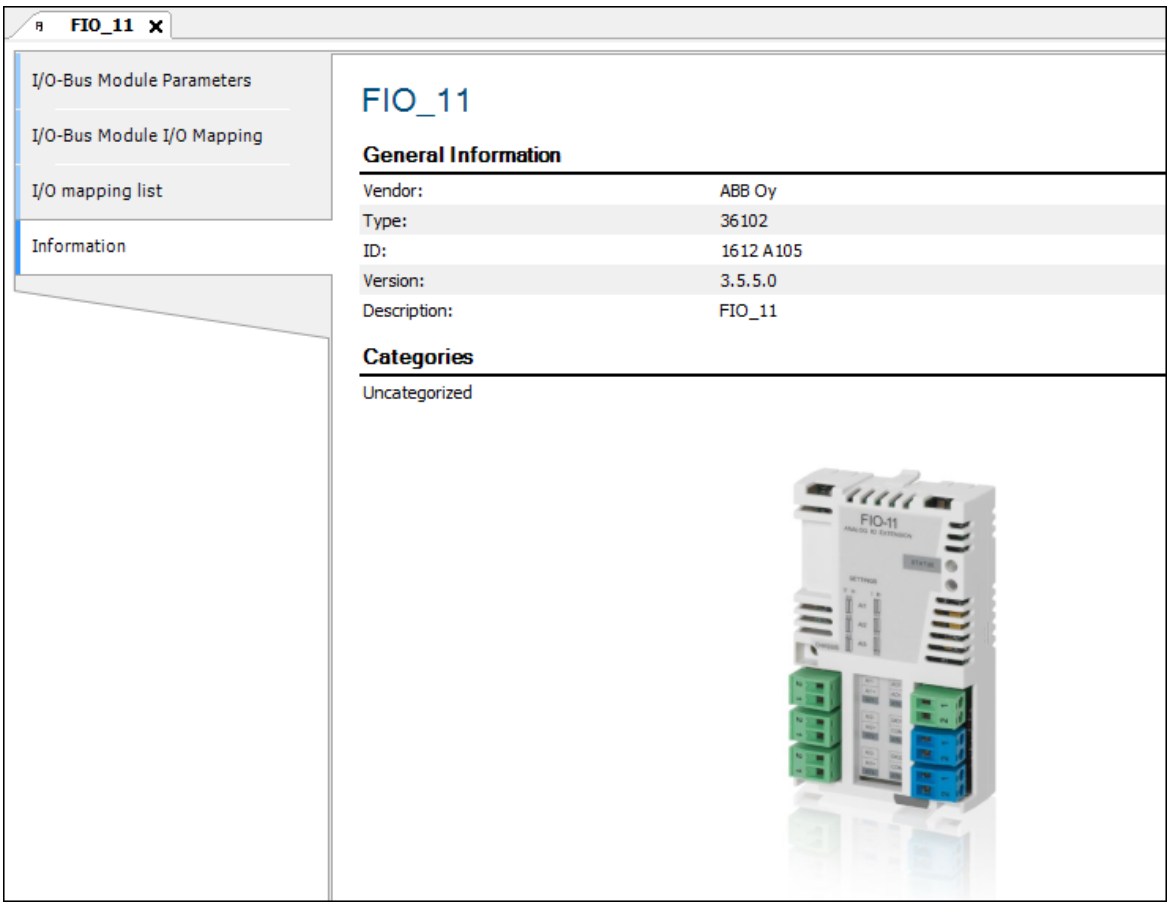
FIO-01 Channel descriptions

Channel	Description
Status	0 = Not active 1 = Initializing state 2 = Active 3 = No communication
Control	0 = Inactivate 1 = Activate FIO-01module
DIOx_FilterTime	Defines the filtering time constant (0.8…100.0 ms). This time is applied for all the filtered inputs (optional).
DIOx_ChDir (x=1-4)	0 = DIO is used as a digital output (default value). 1 = DIO is used as a digital input.
DIOx_Output (x=1-4)	1/0 = ON/OFF status of digital output if channel is used as output (ChDir = 0). The corresponding ON and OFF time delays are applied if they are defined.
DIOx_ONDelay (x=1-4)	Defines activation delay (0.0…300.0 s) applied for digital input/output. This channel is optional.
DIOx_OFFDelay(x=1-4)	Defines deactivation delay (0.0…300.0 s) applied for digital input/output. This channel is optional.
DIOx_FiltInput (x=1-4)	1/0 = ON/OFF status of digital input if channel is used as input (ChDir = 1). Filter time is applied if it is defined. Time delays are never applied.
DIOx_Input (x=1-4)	1/0 = ON/OFF status of digital input if channel is used as input (ChDir = 1). The corresponding ON and OFF time delays are applied if they are defined.

Channel	Description
ROx_Output (x=1-2)	1 = Relay is energized (ON) 0 = Relay is de-energized (OFF)
ROx_ONDelay (x=1-2)	Defines activation delay (0.0…300.0 s) applied for delayed state (optional).
ROx_OFFDelay (x=1-2)	Defines deactivation delay (0.0…300.0 s) applied for delayed state (optional).
ROx_DelayState (x=1-2)	1/0 = ON/OFF status of relay. The corresponding ON and OFF time delays are applied if they are defined.

FIO-11 Module data

You can find the general information of FIO-11 module by clicking on Information tab.



FIO-11 Channel descriptions

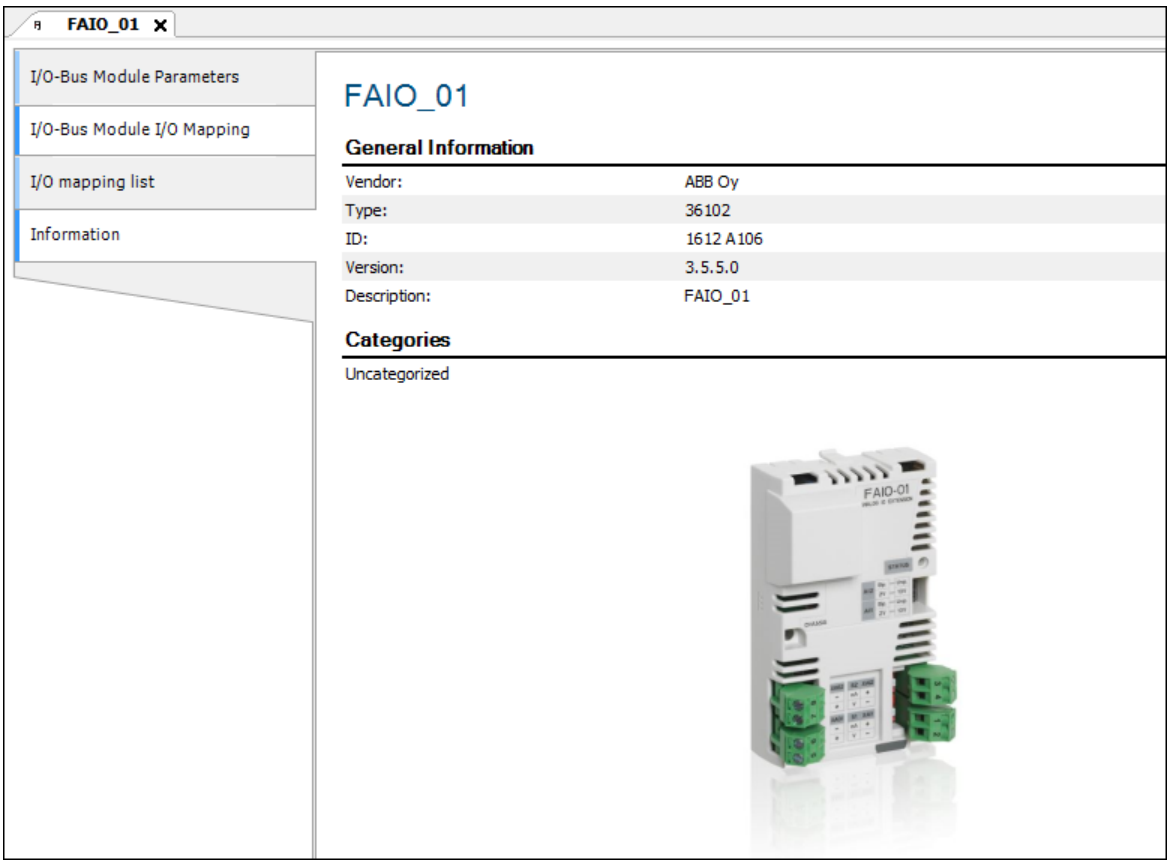
Channel	Description
Status	0 = Not active 1 = Initializing state 2 = Active 3 = No communication
Control	0 = Inactivate 1 = Activates FIO-11 module
DIOx_FilterTime	Defines the filtering time constant (0.8…100.0 ms). This time is applied for all the filtered inputs (optional).
DIOx_ChDir (x=1,2)	0 = DIO is used as a digital output (default value). 1 = DIO is used as a digital input.
DIOx_Output (x=1,2)	1/0 = ON/OFF status of digital output if the channel is used as a output (ChDir = 0). The corresponding ON and OFF time delays are applied if they are defined.
DIOx_ONDelay (x=1,2)	Defines activation delay (0.0…300.0 s) applied for digital input/output. This channel is optional.
DIOx_OFFDelay(x=1,2)	Defines deactivation delay (0.0…300.0 s) applied for digital input/output. This channel is optional.

Channel	Description
DIOx_FiltInput (x=1,2)	1/0 = ON/OFF status of digital input if the channel is used as a input (ChDir = 1). Filter time is applied if it is defined. Time delays are never applied.
DIOx_Input (x=1,2)	1/0 = ON/OFF status of digital input if the channel is used as a input (ChDir = 1). The corresponding ON and OFF time delays are applied if they are defined.
AOx_ForceSel	1 = A forced value is applied for an analog output (optional for testing purposes). 0 = Forcing is not in use.
AO1_FiltTime	Defines the filter time constant (0.000…30.000 s). This time is applied for the filtered analog output. This channel is optional.
AO1_FiltMin	Defines the minimum output value for an analog output (0.000...22.000 mA).
AO1_FiltMax	Defines the maximum output value for an analog output (0.000...22.000 mA).
AO1_FiltMinScaled	Defines the real value (-32768.0…32767.0) that corresponds to the minimum output value (*AO1_FiltMin*). The source value is defined in *AO1_ScaledOut*.
AO1_FiltMaxScaled	Defines the real value (-32768.0…32767.0) that corresponds to the maximum output value (*AO1_FiltMax*). The source value is defined in *AO1_ScaledOut*.
AO1_ScaledOut	Defines the output source value.
AO1_ForceData	Defines the forced value that can be used instead of the output source value *AO1_ScaledOut*. This channel is optional. The forced value (0.000...22.000 mA) is applied for *AO1_Actual* without checking the minimum or maximum output values. Filter time is not applied.
AO1_Actual	The actual analog output value (0.000...22.000 mA). The value is same as in *AO1_Filtered* if forcing in not in use.
AO1_Filtered	The filtered and scaled analog output value (0.000...22.000 mA).
AIx_ForceSel	0 = Forcing is not in use (optional for testing purposes) 1 = Force AI1 to a value of *AI1_ForceData* 2 = Force AI2 to a value of *AI2_ForceData* 3 = Force AI1 to a value of *AI1_ForceData* and AI2 to a value of *AI2_ForceData* 4 = Force AI3 to a value of *AI3_ForceData* 5 = Force AI1 to a value of *AI1_ForceData* and AI3 to a value of *AI3_ForceData* 6 = Force AI2 to a value of *AI2_ForceData* and AI3 to a value of *AI3_ForceData* 7 = Force AI1 to a value of *AI1_ForceData*, AI2 to a value of *AI2_ForceData* and AI3 to a value of *AI3_ForceData*
AIx_Unit (x=1-3)	Unit selection. This setting must match the corresponding hardware setting on the I/O extension module. 2 = V (Volts) 10 = mA (milliamperes)
AIx_Min (x=1-3)	Defines the minimum value for an analog input (-22.000...22.000 mA or V).
AIx_Max (x=1-3)	Defines the maximum value for an analog input (-22.000...22.000 mA or V).
AIx_MinScaled (x=1-3)	Defines the real value (-32768.0…32767.0) that corresponds to the minimum analog input value (*AIx_Min*).
AIx_MaxScaled (x=1-3)	Defines the real value (-32768.0…32767.0) that corresponds to the maximum analog input value (*AIx_Max*).
AIx_FiltTime (x=1-3)	Defines the filter time constant for the analog input (0.000…30.000 s). This time is applied for analog inputs *AIx_Actual* and *AIx_Scaled*. This channel is optional.

Channel	**Description**
AIx_FiltGain (x=1-3)	Selects the hardware filtering time for analog input. This channel is optional. (0 = no filtering, 1 = 125 us, 2 = 250 us, 3 = 500 us, 4 = 1 ms, 5 = 2 ms, 6 = 4ms, 7 = 7,9375 ms).
AIx_ForceData (x=1-3)	Defines the forced value that can be used instead of the true reading of input. This channel is optional. The forced value (-22.000...22.000 mA or V) is applied for *AIx_Actual* without checking minimum or maximum values. Filter time is not applied.
AIx_Actual (x=1-3)	Displays the value of an analog input (-22.000...22.000 mA or V).
AIx_Scaled (x=1-3)	Displays the value of an analog input (-22.000...22.000 mA or V) after scaling.
AIx_Switch (x=1-3)	0 = Unit selection matches the corresponding hardware setting. 1 = Unit selection does not match the corresponding hardware setting.

FAIO-01 Module data

You can find the general information of FAIO-01 module by clicking on **Information** tab.



FAIO-01 Channel descriptions

Channel	Descriptions
Status	0 = Not active 1 = Initializing state 2 = Active (successfully activated by Control) 3 = No communication
Control	0 = Inactivate 1 = Activate FAIO-01 module
AOx_ForceSel	0 = Forcing is not in use output (optional for testing purposes) 1 = *AO1_ForceData* is applied to an analog output *AO1_Actual* 2 = *AO2_ForceData* is applied to an analog output *AO2_Actual* 3 = Both *AO1_ForceData* and *AO2_ForceData* are applied
AOx_FiltTime (x=1,2)	Defines the filter time constant (0.000…30.000 s). This time is applied to the filtered analog output *AOx_Filtered* (optional).
AOx_FiltMin (x=1,2)	Defines the minimum output value to an analog output (0.000...22.000 mA).
AOx_FiltMax (x=1,2)	Defines the maximum output value to an analog output (0.000...22.000 mA).
AOx_FiltMinScaled (x=1,2)	Defines the real value (-32768.0…32767.0) that corresponds to the minimum output value (*AOx_FiltMin*). The source value is defined in *AOx_ScaledOut*.

Channel	**Descriptions**
AOx_FiltMaxScaled (x=1,2)	Defines the real value (-32768.0…32767.0) that corresponds to the maximum output value (*AOx_FiltMax*). The source value is defined in *AOx_ScaledOut*.
AOx_ScaledOut (x=1,2)	Defines the output source value.
AOx_ForceData (x=1,2)	Defines the forced value that can be used instead of the output source value *AOx_ScaledOut*, (optional). The forced value (0.000...22.000 mA) is applied for *AOx_Actual* without checking the minimum or maximum output values. Filter time is not applied.
AOx_Actual (x=1,2)	The actual analog output value (0.000...22.000 mA). The value is same as in *AOx_Filtered* if forcing is not in use.
AOx_Filtered (x=1,2)	The filtered and scaled analog output value (0.000...22.000 mA).
AIx_ForceSel	0 = Forcing is not in use (optional for testing purposes) 1 = Force AI1 to the value of *AI1_ForceData* 2 = Force AI2 to the value of *AI2_ForceData* 3 = Force AI1 to the value of *AI1_ForceData* and AI2 to the value of *AI2_ForceData*
AIx_Unit (x=1,2)	Unit selection. This setting must match the corresponding hardware setting on the I/O extension module. 2 = V (volts) 10 = mA (milliamperes)
AIx_Min (x=1,2)	Defines the minimum value to an analog input (-22.000...22.000 mA or V).
AIx_Max (x=1,2)	Defines the maximum value to an analog input (-22.000...22.000 mA or V).
AIx_MinScaled (x=1,2)	Defines the real value (-32768.0…32767.0) that corresponds to the minimum analog input value (*AIx_Min*).
AIx_MaxScaled (x=1,2)	Defines the real value (-32768.0…32767.0) that corresponds to the maximum analog input value (*AIx_Max*).
AIx_FiltTime (x=1,2)	Defines the filter time constant to an analog input (0.000…30.000 s). This time is applied for the analog inputs *AIx_Actual* and *AIx_Scaled*, (optional).
AIx_FiltGain (x=1,2)	Selects the hardware filtering time to an analog input (optional). (0 = no filtering, 1 = 125 us, 2 = 250 us, 3 = 500 us, 4 = 1 ms, 5 = 2 ms, 6 = 4 ms, 7 = 7,9375 ms).
AIx_ForceData (x=1,2)	Defines the forced value that can be used instead of the true reading of the input (optional). The forced value (-22.000...22.000 mA or V) is applied for *AIx_Actual* without checking minimum or maximum values. Filter time is not applied.
AIx_Actual (x=1,2)	Displays the value of an analog input (-22.000...22.000 mA or V).
AIx_Scaled (x=1,2)	Displays the value of an analog input (-22.000...22.000 mA or V) after scaling.
AIx_Switch (x=1,2)	0 = Unit selection matches the corresponding hardware setting. 1 = Unit selection does not match the corresponding hardware setting.

Fault codes

If the F-series I/O configuration fails, a warning *A7AB Extension I/O configuration failure* is logged in the Event log.

Auxiliary codes	Descriptions
0x1000 – 0x1006	Application related F-series ExtIO configuration file is broken.
0x2000 – 0x2006	Task configuration error in configuration file.
0x2001	No enough communication capacity for requested module type and update times (fast cycle).
0x2002	No enough communication capacity for requested module type and update times (exceeded maximum allowed messages).
0x4000 – 0x4006	DDCS configuration error in configuration file.
0x4003	Unknown task id in DDCS configuration.

9

Libraries

Contents of this chapter

This chapter contains general information of libraries and description of the ABB drives system and standard libraries.

Library types

The following libraries are installed by default in Drive Application Builder for drive programming.

- Default libraries
  - ABB drives system library (AY1LB_System_ACS880_V3_5)
  - ABB drives standard library (AS1LB_Standard_ACS880_V3_5)
- Optional libraries
  - All generic Drive Application Builder IEC libraries (standard and Util) can be installed, but ABB does not guarantee their correct functioning. Note the data type limitations described in chapter *Features (page 47)*.

The Library Manager controls and manages the library usage in the project. Each project has its own Library Manager which is added by default when you create a new project.

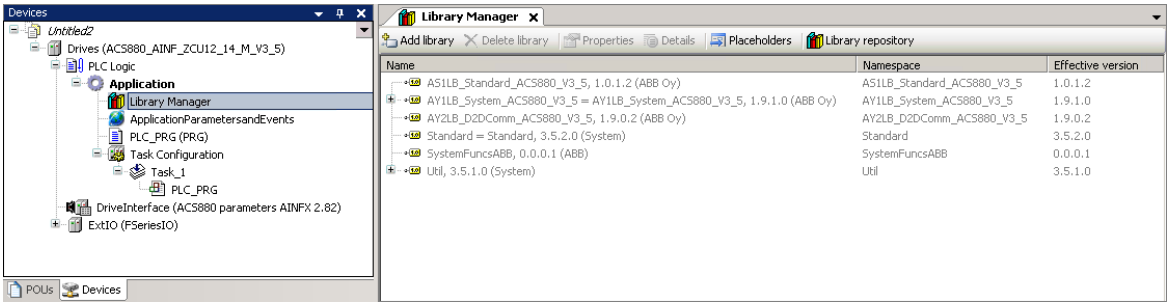


ABB drives standard library contains the most common and useful functions and function blocks to control the drive. All the functions are implemented locally using structured text language. The Drive Application Builder and standard libraries include additional general purpose functions.

ABB drives system library contains all the drive-specific functions to interface the application with the drive firmware and I/O interface. This library has external implementation in the drive system software.

Make sure the drive is installed with the corresponding system library.

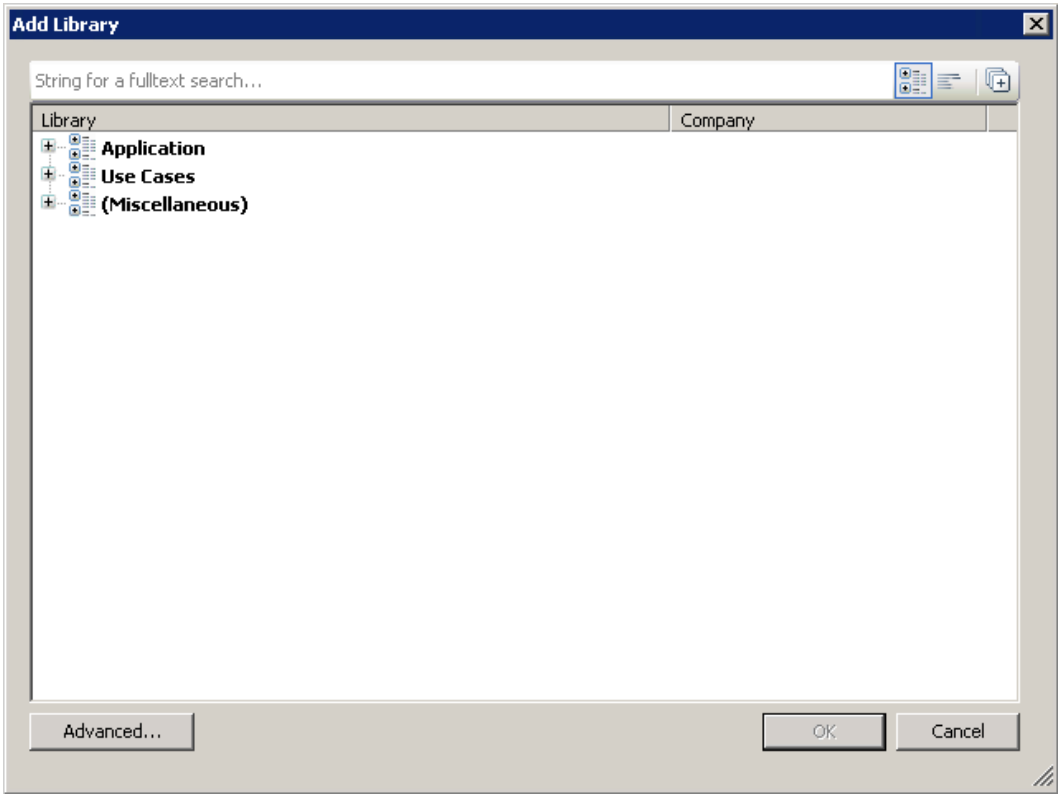
1. In the Drive composer pro, right-click on drive and select **System** info.
2. In the System info screen, click **More**.

Check that the Application System Library displayed in the Drive composer pro has the same library version as the Drive Application Builder project. If the versions are not matching, part of the library may be incompatible.

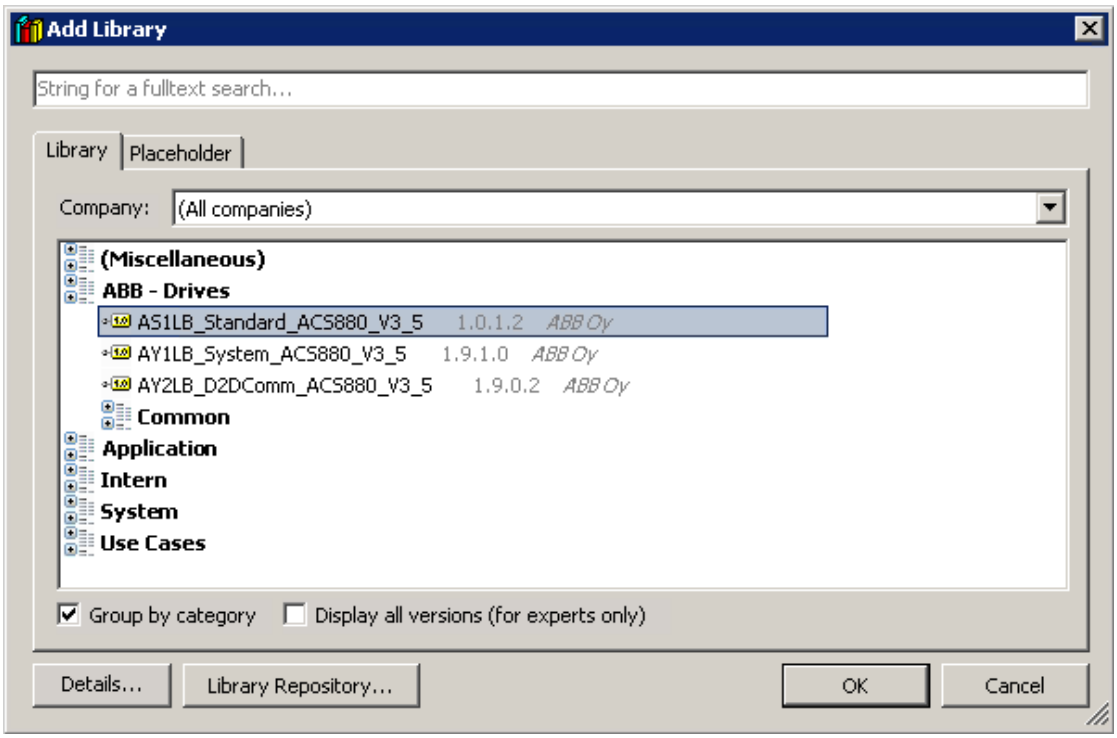
Adding a library to the project

To add a Library Manager (library container) to the project:

1. In the Devices tree, right-click Application and select **Add object**.
2. In the Add object window, select Library Manager and click **Add object**.
3. Double-click **Library Manager**.
   Library Manager window is displayed.
4. Click **Add library** to add the library.
5. In the Add Library dialog, click **Advanced**.



6. Select the required library and click **OK**.



The selected library is added successfully.

Note: To make SFC language programs or functions, the IecSfc system library must be available in the project.

Creating a new library

The application programming environment allows you to create your own libraries to use in the projects. After starting the programming environment, a new library can be created with the New Project dialog.

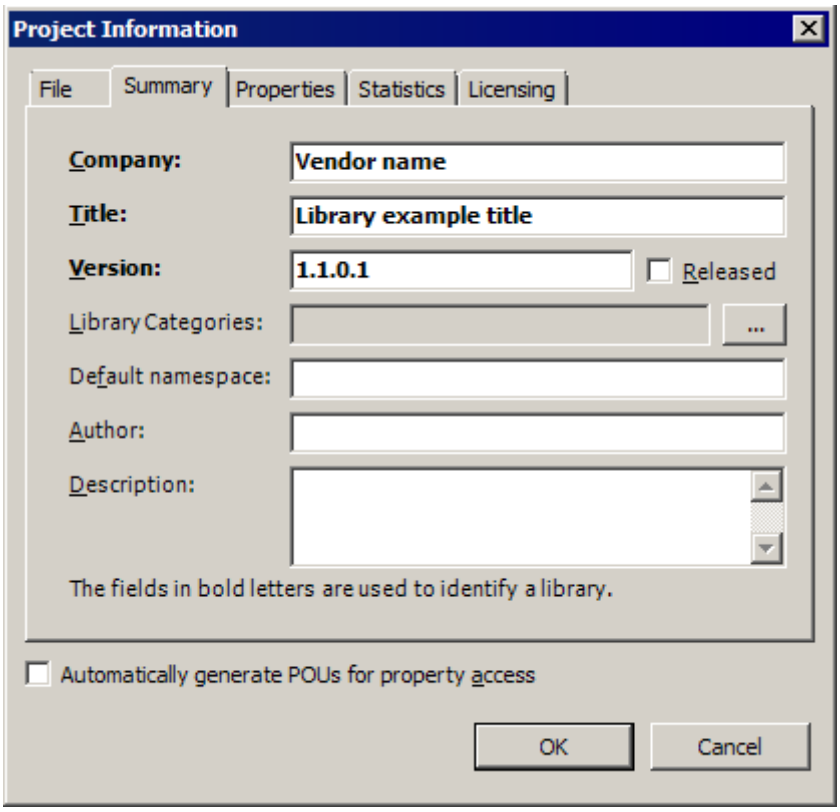
1. In the New Project dialog box, click **Empty project**.
2. Type the library Name and Location and click **OK**.
   The new library is added into the POUs tree.
3. In the View menu, select POUs to add a new POU into the created library.
4. Right-click on project name and select **Add Object** → **POU**.
   Name the POU, for example, POU1.
5. Select the type of the POU, for example, Function Block and the implementation language can be Structured Text (ST) and then click **Add**.
6. Open the created POU and add the following example code into the variables declaration window.

```
FUNCTION_BLOCK POU1

VAR_INPUT
    DI1 : BOOL;
END_VAR

VAR_OUTPUT
    RO1 : BOOL;
END_VAR

VAR
    prev_DI1_value : BOOL;
END_VAR
```

   Add the following example code into the code area:

```
IF DI1 = FALSE AND prev_DI1_value = TRUE THEN
    RO1 := NOT(RO1);
END_IF

prev_DI1_value := DI1;
```

   After the code is added, all library objects must be checked before the library export.
7. In the Build menu select Check all Pool Objects.
8. In the Project menu, select Project Information and fill the information of the created project to use the library in future (company, title and version).



After the information is added, it is possible to install this library directly to the Library Repository.

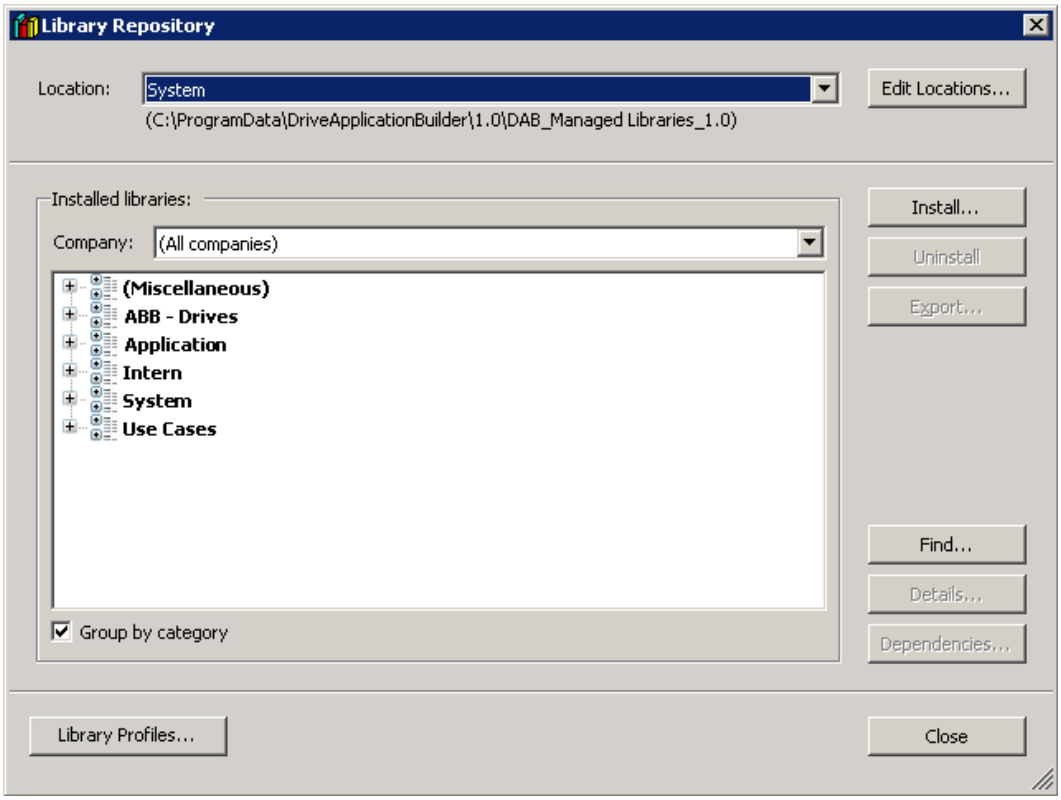
9. In the File menu, you can do the following
   - select Save Project and Install into Library Repository
     Or
   - select Save Project as to save the library as a usual file
     Or
   - select Save Project as Compiled Library to save the library as a compiled library file

   Note: You must use a compiled library file to protect the library source code. The non-compiled library format does not protect the source code.

Installing a new library

To install a new library, follow these steps:

1. In the Drive Application Builder project, double-click **Library Manager**.
2. Click **Add library**.
3. In the Add Library dialog, click **Advanced**.
4. Click **Library Repository**.
5. In the Library Repository window, click **Install**.



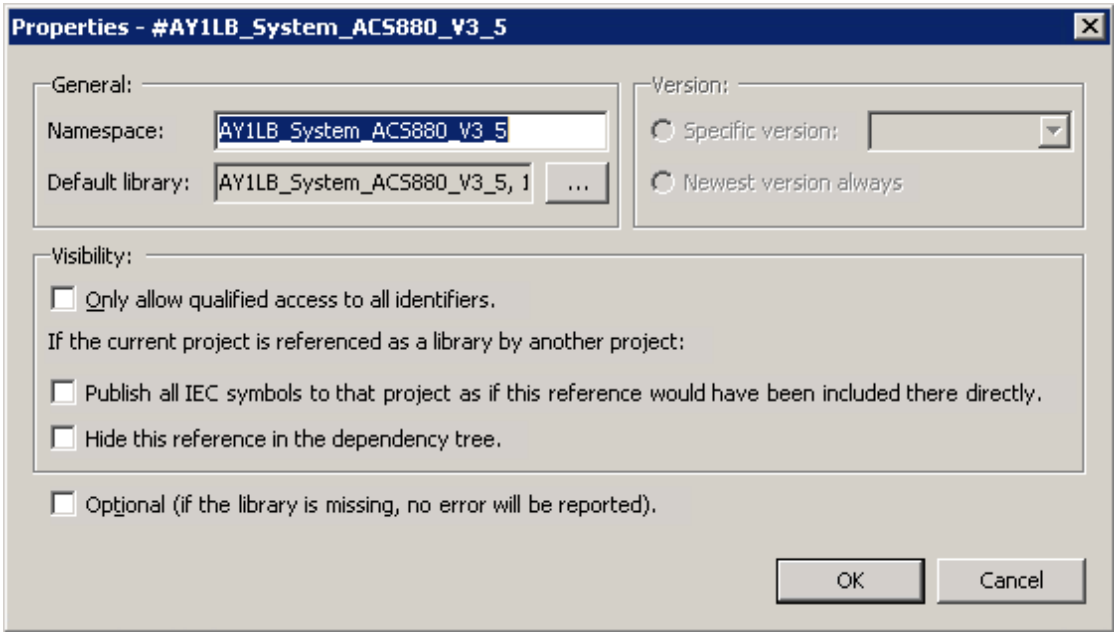
6. Browse/select the required compiled library and click **Open**.
   A new library is installed into the Library Repository and is ready to use in the project.

Managing library versions

Drive Application Builder allows you to use different versions of the selected library according to project requirements.

To change the current effective library version, proceed as follows:

1. Open **Library Manager**.
2. Select the required library and click **Properties**.
3. Select the Specific version in the drop-down list and click **OK**.

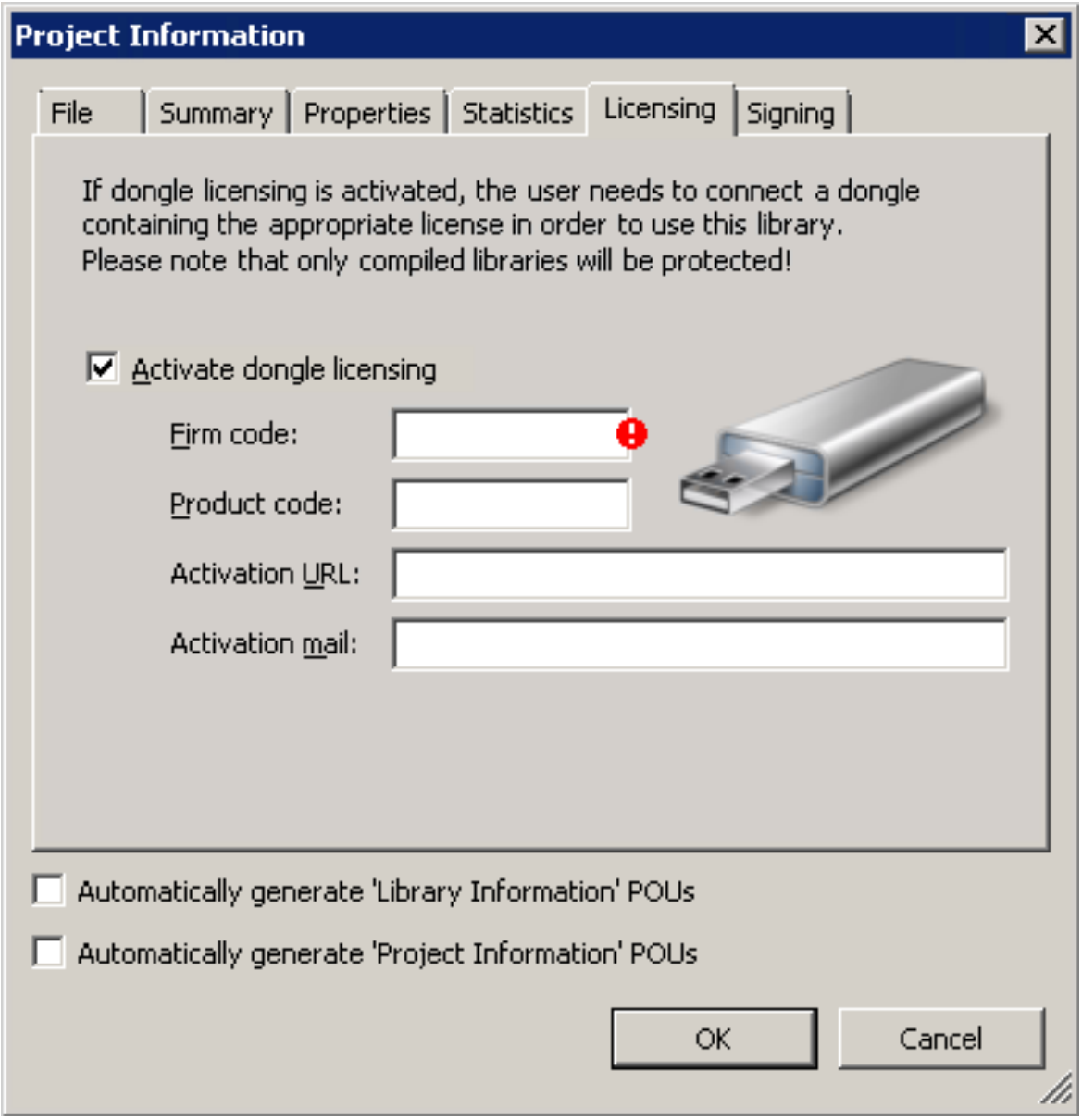


The library version is changed and can be used in the project.
If you want to add a new library version that is not in the Specific version list, first install the version. See section *Libraries (page 107)*.

Configuring a library with WIBU license

In Drive Application Builder, you can configure a library with WIBU license.

1. In Drive Application Builder main menu, go to **Project** → **Project Information**.
2. In the **Licensing** tab, do the following:
   - Enable **Activate dongle licensing** check-box and add the Firm code, Product code, Activation URL and Activation mail.
   - Click **OK**.



The license protection can be used only for compiled libraries and it is necessary that the given license is already present on the dongle.

10

Practical examples and tips

Contents of this chapter

This chapter gives practical examples and tips on working with Drive Application Builder.

Solving communication problems

Follow the instructions below when the scan network does not find any drives.

1. Check the communication settings.
2. In Windows **Computer Management** → **Device Manager**, check that the communication port is correctly installed.
3. If the USB Serial Port (COMX) is not displayed under Device Manager, check that the corresponding USB/communication port driver is installed.
4. Select Ctrl + shift + esc → **Processes** to check that the OPC server (DriveDA.exe) has started in Windows Task Manager.
5. Check that the Drive composer pro (Drive OPC) finds the connection to the drive.

   Note: You must allow Drive Application Builder to share communication with Drive composer pro.

For details on how to allow Drive Application Builder to share communication with Drive composer pro, see chapter *Getting started (page 19)*.

Follow the instructions below when the communication fails between Drive Application Builder/Drive composer pro and drive.

- Check that the control panel has the latest firmware version
- Check the Driver data

Note: The next panel driver version is not known. For version details, refer the corresponding ACS880 drive software release notes or contact your ABB representative.

Solving other problems

- **How to prevent unauthorized access to an application that is running in the drive?**
  A compiled project as well as the downloaded source code can be password protected. You can make a backup copy of the protected application. The backup copy is encrypted and you need a password for downloading or executing the copied application. The IEC function libraries and projects can be protected as well by means of Drive Application Builder.
- **What to do when stack overflow fault 6487 occurs?**
  - **If the stack overflow fault 6487 occurs, then the number of the local variables inside a function is too large. Unfortunately, the limit of the local variables are relatively small. The stack usage is high especially if there are, for example, division operands inside the EXPT function.**
  - **Also if the division function divider is zero (an exceptional case), the stack usage is high.**

  Do not make large functions. Try to make a compact function with a limited number of variables (40 REAL). If the function is too large, change some of the local variables to global variables (use, for example, multiple global variable lists GVL to group variables by functions). Consider to use function blocks or program modules instead of functions.
- **How to optimize the memory usage of the drive application? The code memory of the application is running out. How to optimize the program?**
  The drive application programming environment has relatively limited memory and execution capacity. There are a couple of tips to minimize the program code:
  - Use functions as much as possible.

    Note: If there are many variables inside the function, the risk of stack overflow increases.
  - Try to design the application so that you do not need to create many instances of large function blocks. Instead of function blocks use programs or functions.
  - Use DriveInterface to access drive parameters instead of the parameter read/write functions.
- **How to solve the problem causing error message “Creating boot application failed: Adding Application Parameters & Groups to UFF generator: XmlDeserializationFailed”?**
  The problem is related to Application parameters and events module.
  - Check that all Value pointer, Bit pointer and Plain value list type of parameters have the correct Selection List.
  - Check that the Bit list (16 bit) parameters do not have same Bit names (English) multiple times (for example, text Bit_Handle_0 occurs twice).
  - Check the tool message box for details.

11

Unsupported features

Contents of this chapter

This chapter lists the features that are not supported for drives with standard drive application programming V3.

Unsupported features

ABB drives do not support the following standard drive application programming V3 features.

- Persistent variable type is not supported. In case the variable is saved over power cycle, retain variable is used. Also, the user defined drive parameter can be created to save value of the variable.
- Target-based tracing. You can use the Monitor feature in Drive composer pro. See *Drive composer user's manual* (3AUA0000094606 [English]).
- Some data types are not supported.
- The number of program execution tasks are limited to 4. One of the task is a pre task which is executed only once after power up. Other tasks are cyclically executed.
- Program code simulation is not supported.
- Target based visualization is not supported.

12

ABB drives system library

Contents of this chapter

This chapter contains detailed information of the function blocks of the ABB drives system library (AS1LB_Standard_ACS880_V3_5).

Overview

The ABB drives system library is intended to use with the ACS880 drives. It provides event, parameter read/write and program time level function blocks for application program in the Drive Application Builder environment. The description of the features in this document is based on the ABB drives system library version 1.9.1.0.

Using Drive composer pro System info, check that the drive is installed with the corresponding system library. In the System info, the system library version is located under the **Products**/**More** view. The system library versions must be similar in the drive and the application program project.

Function blocks of the system library

Function block name	Description
Event function blocks	
EVENT	Send the application event
ReadEventLog	Read the drive's faults and warnings
Parameter change function blocks	
PAR_UNIT_SEL	Changes the unit of a parameter
PAR_SCALE_CHG	Changes the parameter scaling attributes
PAR_LIM_CHG_DINT	Changes the limits of a parameter in DINT data format
PAR_LIM_CHG_REAL	Changes the limits of a parameter in REAL data format
PAR_LIM_CHG_UDINT	Changes the limits of a parameter in UDINT data format
PAR_DEF_CHG_DINT	Changes the default values of a parameter in DINT data format
PAR_DEF_CHG_REAL	Changes the default values of a parameter in REAL data format
PAR_DEF_CHG_UDINT	Changes the default values of a parameter in UDINT data format
PAR_DISP_DEC	Changes the decimal display of a parameter
PAR_REFRESH	Notifies PC tools and panel of any parameter attribute changes
Parameter protection	
PAR_PROT	Protects individual parameters
PAR_GRP_PROT	Protects a parameter group
Parameter read function blocks	
ParReadBit	Read the value of a bit in a packed-Boolean-type parameter
ParRead_INT	Read the value of an INT/DINT/REAL type parameter
ParRead_DINT	Read the value of a DINT/INT type parameter
ParRead_REAL	Read the value of a REAL type parameter
ParRead_UDINT	Read the value of a UDINT/UINT type parameter
Parameter write function blocks	
ParWriteBit	Write the value to a bit of a packed-Boolean-type parameter
ParWrite_DINT	Write the value to a REAL/DINT/INT type parameter
ParWrite_INT	Write the value to an INT/DINT/REAL type parameter
ParWrite_REAL	Write the value to a REAL type parameter
ParWrite_UDINT	Write the value to an UDINT/UINT type parameter
Pointer parameter read function blocks	
ParRead_BitPTR	Read the pointed bit value from a bit pointer type parameter
ParRead_ValPTR_DINT	Read the pointed DINT/INT value from a value pointer type parameter

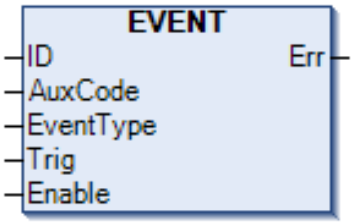
Function block name	Description
ParRead_ValPTR_REAL	Read the pointed REAL value from a value pointer type parameter
ParRead_ValPTR_UDINT	Read the pointed UDINT/UINT value from a value pointer type parameter
GetPtrParConf	Read the source parameter settings. Source parameter must be a value pointer, bit pointer or formatted number
Set pointer parameter function blocks	
ParSet_BitPTR_IEC	Set a bit pointer parameter to point to a bit type IEC variable
ParSet_ValPTR_IEC_DINT	Set a value pointer parameter to point to a DINT type IEC variable
ParSet_ValPTR_IEC_REAL	Set a value pointer parameter to point to a REAL type IEC variable
ParSet_ValPTR_IEC_UDINT	Set a value pointer parameter to point to an UDINT type IEC variable
ParSet_BitPTR_Par	Set a bit pointer parameter to point to a bit of a packed Boolean parameter
ParSet_ValPTR_Par	Set a value pointer parameter to point to a value parameter
System time function blocks	
SYS_TIME	Shows the previously set system data, time (broken time) and source
SYS_TIME_UDNIT	Shows the previously set system data, time (raw time) in native format and source
Task time level function block	
UsedTimeLevel	Show time level (ms) of the program where the function block is located

Event function blocks

EVENT

Summary

The application event function block is used to trigger a predefined event (fault/warning/pure) from the IEC code. The event is registered to drive event logger.



Connections

Inputs

Name	Type	Value	Description
ID	WORD	0xE100.. 0xE2FF	Identification of the event (constant, cannot be changed on run time). This is a unique value of the event. You can find the supported values in the ApplicationParametersAndEvent tool. A certain range is reserved for each application event type. Faults: 0xE100…E1FF Warnings: 0xE200.. 0xE2FF
AuxCode	DWORD	ANY	The auxiliary code that you can set freely (constant).
Event-Type	WORD	1, 2	Type of the event (constant, cannot be changed on run time). Supported event types: Fault = 1, Warning = 2, Pure = 8 (Notice is not supported).
Trig	BOOL	T/F	The high level (TRUE) of this pin sends/activates the event, if Enable is set to TRUE. Warning is deactivated automatically, when Trig is decreased. To clear the fault, give the reset command.
Enable	BOOL	T/F	Enable/disable event sending.

Outputs

Name	Type	Value	Description
Err	WORD	ANY	The value is typically 0x0000. 0x0001 = Not used 0x0002 = Event is not user-defined event 0x0003 = Event type error 0x0004 = Event ID type error 0x0005 = Not used 0x0006 = Unknown event type

Description

You can configure an application event with the ApplicationParametersandEvents in Drive Application Builder. (See chapter *Application parameters and events (page 77)*). This tool defines the ID and the event text (description).

Drive Application Builder supports the following event types: Fault, Warning and Pure.

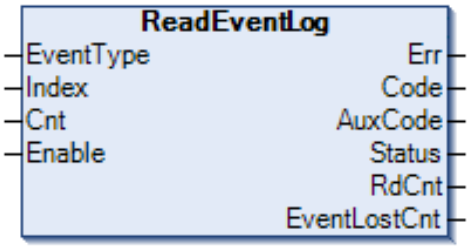
The event ID, text, auxiliary code, time and operation data is registered into the drive event logger. The application events can be shown using the ACS-AP-x control panel and Drive composer tools, or using the ReadEventLog block on the application level. A fault can be reset, for example, using the control panel or Drive composer pro tool.

Note: The current firmware supports execution of three event functions in the same task cycle. If there are more event functions, do not enable all of them at the same time.

ReadEventLog

Summary

ReadEventLog is a special block for reading faults and warnings from the drive event system. The block does not read events or use the drive event or fault loggers. Instead it gets the events straight from the event system itself.



The purpose of the block is to forward drive events, for example, to external systems, like automation user interfaces.

Connections

Inputs

Name	Type	Value	Description
EventType	UINT	0	Not used. The block returns the drive's faults and warnings. The value can be set to 0.
Index	UINT	0	Not used. The value can be set to 0.
Cnt	UINT	0…6	Number of the wanted events at a time (0…6).
Enable	BOOL	T/F	Enable/disable event sending.

Outputs

Name	Type	Value	Description
Err	UINT	N/A	Not used.
Code	Array of UINT[10]	Any of allowed events codes	Event code (ID). The block supports maximum 6 events at a time.
AuxCode	Array of UINT[10]	ANY	Auxiliary code of the event.
Status	Array of UINT[10]	ANY	Status of the event. 1 = Event is activated. 2 = Event is deactivated. 3 = Acknowledgement requested. 4 = Event is reactivated (warnings). 5 = All faults are deactivated.
RdCnt	UINT	0…6	The number of the get/read events at a time. Maximum 6 RdCnt value = 0 indicates that there are no new events.
EventLostCnt	UINT	ANY	The number of the lost events (for monitoring).

Note: The current firmware supports execution of three event functions in the same task cycle. If there are more event functions, do not enable all of them at the same time.

It is recommend to use event blocks only on the tasks that has the cycle time setting higher than 50ms.

Description

The block packs the event *Code*, *AuxCode* and *Status* to vectors that the user can read. The block does not sort faults and warnings from each other. The first event in the vector is the oldest one.

The block returns the maximum Cnt number of events in each execution cycle depending on how many events exist at this time on the drive. *RdCnt* indicates how many events are got in each execution cycle. The vectors and *RdCnt* are updated in every execution cycle if new events exist. For this reason, only the value of *RdCnt* matters when reading the event data from vectors. The older events are overwritten by the newer ones.

Example:

In the first execution cycle, the user reads 2 events, for example, events 11, 12 (*RdCnt* = 2). Both are valid. 12 is the last one.

In the second execution cycle, the user reads 1 event, for example, 21 (*RdCnt* = 1).

Now values 21, 12 can be seen in the Code vector, but because *RdCnt* is 1, only the first value is valid (21). (12 read in the previous cycle.)

Vectors are cleared only on the falling edge of the Enable pin.

EventLostCnt indicates the number of the lost events. The value should be 0. In the opposite case, the reason can be too slow execution cycle of this block.

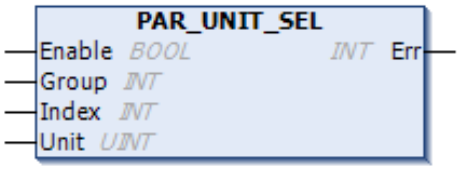
Note: The execution cycle of this block is slow. To optimize the application resources, it is recommended to use only one instance of this block.

Parameter change function blocks

PAR_UNIT_SEL

Summary

PAR_UNIT_SEL block enables to change the unit of a parameter from the IEC application. If one parameter of the family parameter is changed using this block, the change applies to all other parameters of that parameter family.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables unit change at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Unit	UNIT	128...255	Unit selection

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the unit change of a parameter. *Group* and *Index* define the parameter to be changed and *Unit* defines the unit of the parameter. The unit strings and corresponding codes are defined in the Drive Application Builder, ApplicationparameterandEvents manager (APEM). Using this function block, the units in the range of 128 to 255 can be changed.

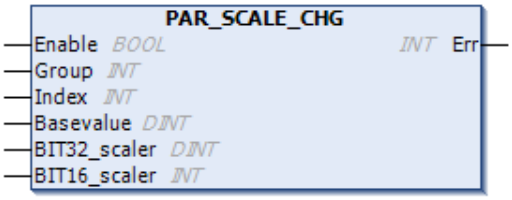
Note: Use only the units defined in APEM. Selecting undefined units are not notified by the Err output.

Err returns an error code if there is an error during a unit change, for example, the unit for change is beyond the selection range. If the unit selection and change operation is successful, *Err* returns a 0.

PAR_SCALE_CHG

Summary

PAR_SCALE_CHG block enables changing the parameter scaling attributes from the IEC application. Initial scaling values are defined in the Parameter family settings.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables scale change at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Basevalue	DINT	128...255	Scales internal value to external 32 or 16 bit interface. Used as divider
BIT32_scaler	DINT	ANY	Scaling factor for external 32 bit interface in panel (ACS-AP-I), DriveComposer and fieldbus interface. The value is used as a multiplier.
BIT16_scaler	INT	ANY	Scaling factor for external 16 bit interface for fieldbus interface. The value is used as a multiplier.

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block enables to change the parameter scaling factor that scales the internal value for DriveComposer tool, ACS-AP-I panel and fieldbus interface. The initial values of the scaling factors are defined in ApplicationparameterandEvents manager (APEM) for all user parameters. The changed parameter scaling applies to all parameters of a specific family (scaling) defined in APEM.

The rising edge of *Enable* input implies the parameter scaling change. *Group* and *Index* define the parameter to be changed. The *Basevalue* scales the internal value to external 32 or 16 bit interface.

The *BIT32_scaler* and *BIT16_scaler* are used as scaling interfaces.

The *Err* output returns an error code if there is an error during the scaling change operation. If the scaling changes are successful, *Err* returns a 0.

External 32-bit scaling

The external 32-bit scaling is used by (ACS-AP-I), Drive Composer and PLC over fieldbus adapter. If the parameter type is REAL, the number of decimals influence the scaling defined in

ApplicationparametersandEvents manager or the PAR_DISP_DEC block.

If external value is requested as 32-bit integer, the internal float is scaled to external float with the same scaling factor and then converted to 32 bit integer with extra numbers for decimal values, depending on the display format of decimals. For example: The value 1.23456 is displayed as 1.235 if the display format is 3 decimals.

Scaling formula:

$$Externalvalue(32bit) = \frac{BIT32scaler * 10^{Decimals}}{Basevalue} * IECprogramvariable(internalvalue)$$

External 16-bit scaling

The external 16-bit scaling is used only for fieldbus interface to fit internal value with higher number of bits to the 16-bit scale. The 16-bit external value uses its own scaling factor with no display format for decimals.

Scaling formula:

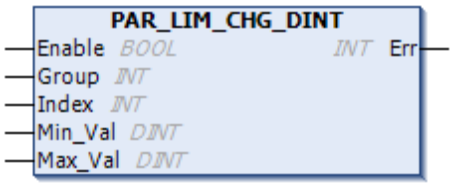
$$Externalvalue(16bit) = \frac{BIT16scaler}{Basevalue} * IECprogramvariable(internalvalue)$$

Parameter limit change

PAR_LIM_CHG_DINT

Summary

The PAR_LIM_CHG_DINT block enables to change minimum and maximum values (in DINT data format) of a parameter from the IEC application. The changes in the limit values apply to all parameters belonging to same parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing parameter limits at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Min_Val	DINT	ANY	New minimum value in DINT data format
Max_Val	DINT	ANY	New maximum value in DINT data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter limit values. *Group* and *Index* define the parameter to be changed. The *Min_Val* and *Max_Val* are used to set the new minimum and maximum values of the parameter respectively.

Note: Make sure that the following conditions are met while defining the minimum and maximum values:

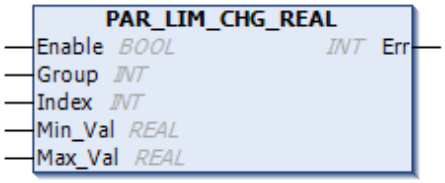
- The *Max_Val* must be greater than *Min_Val*.
- The *Min_Val* must be lesser than *Max_Val*.
- *Min_Val* must not be equal to *Max_Val*.

Err returns an error code if there is an error during the limits change operation, for example, the new limits are beyond the range. If the change operation is successful, *Err* returns a 0.

PAR_LIM_CHG_REAL

Summary

The PAR_LIM_CHG_REAL block enables changing the minimum and maximum values (in REAL data format) of the parameter from the IEC application. The changes in the limit values apply to all parameters belong to the same parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing parameter limits at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Min_Val	REAL	ANY	New minimum value in REAL data format
Max_Val	REAL	ANY	New maximum value in REAL data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter limit values. *Group* and *Index* define the parameter to be changed. The *Min_Val* and *Max_Val* are used to set the new minimum and maximum values of the parameter respectively.

Note: Make sure that the following conditions are met while defining the minimum and maximum values:

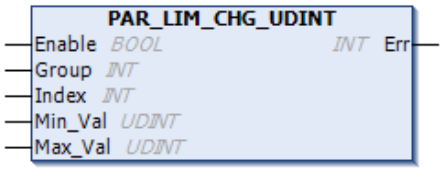
- *Max_Val* must be greater than *Min_Val*
- *Min_Val* must be lesser than *Max_Val*
- *Min_Val* must not be equal to *Max_Val*

Err returns an error code if there is an error during the limits change operation, for example, the new limits are beyond the range. If the change operation is successful, Err returns a 0.

PAR_LIM_CHG_UDINT

Summary

The PAR_LIM_CHG_UDINT block enables changing the minimum and maximum values (in UDINT data format) of a parameter from the IEC application. The changes in the limit values apply to all parameters belong to the same parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing parameter limits at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Min_Val	UDINT	ANY	New minimum value in UDINT data format
Max_Val	UDINT	ANY	New maximum value in UDINT data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter limit values. *Group* and *Index* define the parameter to be changed. The *Min_Val* and *Max_Val* are used to set the new minimum and maximum values of the parameter respectively.

Note: Make sure that the following conditions are met while defining the minimum and maximum values:

- *Max_Val* must be greater than *Min_Val*
- *Min_Val* must be lesser than *Max_Val*
- *Min_Val* must not be equal to *Max_Val*

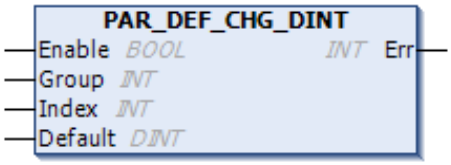
Err returns an error code if there is an error during the limits change operation, for example, the new limits are beyond the range. If the change operation is successful, *Err* returns a 0.

Parameter default value change

PAR_DEF_CHG_DINT

Summary

The PAR_DEF_CHG_DINT block enables changing the default values (in DINT data format) of a parameter from the IEC application. The value changes apply to all parameters of that specific parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing the default value of a parameter at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Default	DINT	ANY	New default value in DINT data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter default values. *Group* and *Index* define the parameter to be changed. The input Default is used to set the new default value of the parameter.

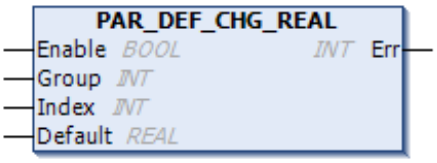
Note: Define a default value within the minimum and maximum value.

Err returns an error code if there is an error during the change operation. If the default value change operation is successful, *Err* returns a 0.

PAR_DEF_CHG_REAL

Summary

The PAR_DEF_CHG_REAL block enables changing the default values (in REAL data format) of a parameter from the IEC application. The value changes apply to all parameters of that specific parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing the default value of a parameter at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Default	REAL	ANY	New default value in REAL data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter default values. *Group* and *Index* define the parameter to be changed. The input Default is used to set the new default value of the parameter.

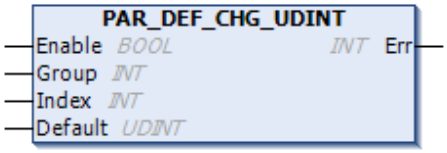
Note: Define a default value within the minimum and maximum value.

Err returns an error code if there is an error during the change operation. If the default value change operation is successful, *Err* returns a 0.

PAR_DEF_CHG_UDINT

Summary

The PAR_DEF_CHG_UDINT block enables changing the default values (in UDINT data format) of a parameter from the IEC application. The value changes apply to all parameters of that specific parameter family defined in APEM.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables changing the default value of a parameter at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Default	UDINT	ANY	New default value in UDINT data format

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the changed parameter default values. *Group* and *Index* define the parameter to be changed. The input Default is used to set the new default value of the parameter.

Note: Define a default value within the minimum and maximum value.

Err returns an error code if there is an error during the change operation. If the default value change operation is successful, *Err* returns a 0.

Parameter decimal display

PAR_DISP_DEC

Summary

PAR_DISP_DEC block enables changing the number of displayed decimals of a parameter from the IEC application. If one parameter of the family parameter is changed using this block, then the change applies to all the other parameters of that parameter family.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables decimal display change at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Decimals	UINT	128...255	Number of decimals to display

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

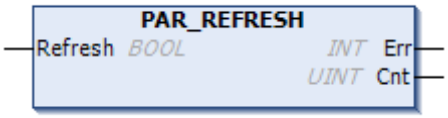
The rising edge of *Enable* input implies the decimal display change of a parameter. *Group* and *Index* define the parameter to be changed and the input Decimals defines the number of decimal values to display. If the parameter is in REAL data format, then the value is scaled for fieldbus interface by scaling factor $10^{(decimals)}$.

Err returns an error code if there is an error during a unit change, for example, the unit for change is beyond the selection range. If the unit selection and change operation is successful, *Err* returns a 0.

PAR_REFRESH

Summary

PAR_REFRESH block notifies PC tools and panel of any parameter attribute changes.



Connections

Inputs

Name	Type	Value	Description
Refresh	BOOL	T/F	Enables refresh at the rising edge

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output
Cnt	UINT	ANY	Counts the number of refresh activation

Description

The rising edge of *Refresh* input notifies any parameter changes to PC tools and panel.

WARNING!
Every time you activate the Refresh input in Drive Application Builder, a notification appears in Drive Composer prompting to refresh the parameters. Click **OK** to apply the parameter changes.

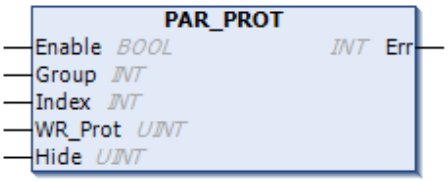
Err returns an error code if the parameter protection is applied successfully, *Err* returns a 0. The output Cnt increments at every activation of the input Refresh.

Parameter protection

PAR_PROT

Summary

PAR_PROT block is used to protect individual parameters. The block enables write protection and hides flags dynamically from the IEC application. The changes do not apply to any other parameter of the specific family.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables protection change at the rising edge
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
WR_Prot	UINT	ANY	Applies write protection 0 = No protection 1 = Human WP [Drive Composer (Pro/Entry) and ACS-AP-I/ACS-AP-S control panel]
Hide	UINT	ANY	Hides flags 0 = No protection 1 = Human WP [Drive Composer (Pro/Entry) and ACS-AP-I/ACS-AP-S control panel]

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the protection change of a parameter. *Group* and *Index* define the parameter to be changed. The inputs *WR_Prot* and Hide define the parameter for write protection and parameter to hide respectively.

Err returns an error code if there is an error during a parameter protection change. If the parameter protection is applied successfully, *Err* returns a 0.

PAR_GRP_PROT

Summary

PAR_GRP_PROT block is used to protect a parameter group. This block enables write protection and hides flags dynamically from the IEC application.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables protection at the rising edge
Group	INT	ANY	Parameter group
WR_Prot	UINT	ANY	Applies write protection 0 = No protection 1 = Human WP [Drive Composer (Pro/Entry) and ACS-AP-I/ACS-AP-S control panel]
Hide	UINT	ANY	Hides flags 0 = No protection 1 = Human WP [Drive Composer (Pro/Entry) and ACS-AP-I/ACS-AP-S control panel]

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The rising edge of *Enable* input implies the protection change of a parameter group. *Group* defines the group to be changed. The inputs *WR_Prot* and Hide define the parameter group to be write protected and hidden.

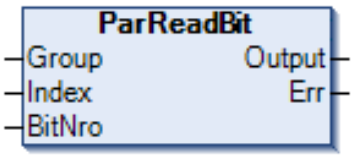
Err returns an error code if there is an error during a protection change. If the parameter group protection is applied successfully, *Err* returns a 0.

Parameter read function blocks

ParReadBit

Summary

ParReadBit reads the value of a bit in a packed Boolean type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
BitNro	INT	ANY	Bit number

Outputs

Name	Type	Value	Description
Output	BOOL	T/F	Output value
Err	INT	ANY	Error output

Description

The function block reads the value of a bit in a packed Boolean type parameter. *Group* and *Index* define the parameter to be read and *BitNro* defines the number of the bit. The value of the bit read is returned from *Output*.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_INT

Summary

ParRead_INT reads the value of a INT/DINT/REAL type parameter.

Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	INT	ANY	Output value
Err	INT	ANY	Error output

Description

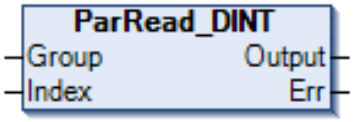
The function block reads the value of a DINT or INT type parameter. *Group* and *Index* define the parameter to be read. The value of the parameter is returned from Output. The type of output is *INT* even if the parameter to be read is of the DINT/REAL type.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_DINT

Summary

ParRead_DINT reads the value of a DINT/INT type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	DINT	ANY	Output value
Err	INT	ANY	Error output

Description

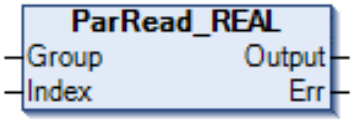
The function block reads the value of a DINT or INT type parameter. *Group* and *Index* define the parameter to be read. The value of the parameter is returned from *Output*. The type of *Output* is DINT even if the parameter to be read is of the INT type.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_REAL

Summary

ParRead_REAL reads the value of a REAL type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	REAL	ANY	Output value
Err	INT	ANY	Error output

Description

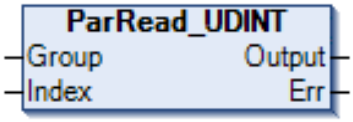
The function block reads the value of a REAL type parameter. *Group* and *Index* define the parameter to be read. The value of the parameter is returned from *Output*.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_UDINT

Summary

ParRead_UDINT reads the value of a UDINT/UINT type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	UDINT	ANY	Output value
Err	INT	ANY	Error output

Description

The function block reads the value of UDINT or UINT type parameter. *Group* and *Index* define the parameter to be read. The value of the parameter is returned from *Output*. The type of the output is UDINT even if the parameter to be read is of the UINT type.

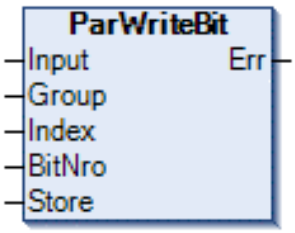
Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, Er*r* returns a 0.

Parameter write function blocks

ParWriteBit

Summary

ParWriteBit writes a value to a bit of the packed Boolean type parameter.



Connections

Inputs

Name	Type	Value	Description
Input	BOOL	T/F	Input value
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
BitNro	INT	ANY	Bit number
Store	BOOL	T/F	Store input

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

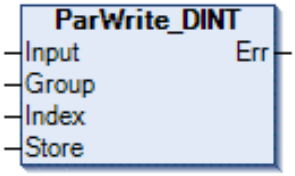
The function block writes the value of Input into a selected bit of a packed Boolean type parameter. *Group* and *Index* define the parameter to be written and *BitNro* define the number of the bit. Store defines if the current written value of the parameter is stored to the flash memory. During the power up of the drive, the value of the parameter is set to the latest stored value.

Err returns an error code if there is an error during the write operation, for example, the parameter is not found or it is a parameter of a wrong type. If the write operation is successful, *Err* returns a 0.

ParWrite_DINT

Summary

ParWrite_DINT writes a value to a REAL/DINT/INT type parameter.



Connections

Inputs

Name	Type	Value	Description
Input	DINT	ANY	Input value
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	Store input

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

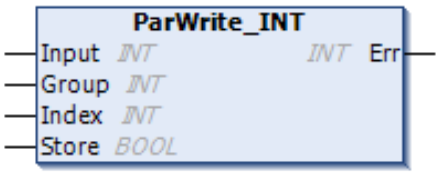
The function block writes the value of Input into a selected DINT or INT type parameter. The type of the Input is DINT even if the parameter to be written is of the INT/REAL type. *Group* and *Index* define the parameter to be written. Store defines if the current written value of the parameter is stored to the flash memory. During the power-up of the drive, the value of the parameter is set to the latest stored valued.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParWrite_INT

Summary

ParWrite_INT writes a value to an INT/DINT/REAL type parameter.



Connections

Inputs

Name	Type	Value	Description
Input	INT	ANY	Input value
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	Store input

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block writes the value of Input into a selected INT type parameter. The type of the Input is INT even if the parameter to be written is of the DINT/REAL type. In case of application parameter, select 16-bit interface support.

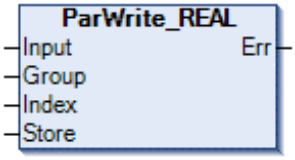
Group and *Index* define the parameter to be written. Store defines if the current written value of the parameter is stored to the flash memory. During the power up of the drive, the value of the parameter is set to the latest stored value.

Err returns an error code if there is an error during the write operation, for example, the parameter is not found or it is a parameter of a wrong type. If the write operation is successful, *Err* returns a 0.

ParWrite_REAL

Summary

ParWrite_REAL writes a value to a REAL type parameter.



Connections

Inputs

Name	Type	Value	Description
Input	REAL	ANY	Input value
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	Store input

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

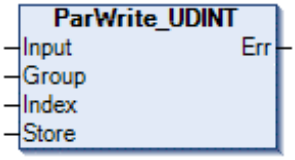
The function block writes the value of Input into a selected REAL type parameter. *Group* and *Index* define the parameter to be written. Store defines if the current written value of the parameter is stored to the flash memory. During the power up of the drive, the value of the parameter is set to the latest stored value.

Err returns an error code if there is an error during the write operation, for example, the parameter is not found or it is a parameter of a wrong type. If the write operation is successful, *Err* returns a 0.

ParWrite_UDINT

Summary

ParWrite_UDINT writes a value to a UDINT/UINT type parameter.



Connections

Inputs

Name	Type	Value	Description
Input	UDINT	ANY	Input value
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	Store input

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block writes the value of Input into a selected UDINT or UINT type parameter. The type of Input is UDINT even if the parameter to be written is of the UINT type. *Group* and *Index* define the parameter to be written. Store defines if the current written value of the parameter is stored to the flash memory. During the power up of the drive, the value of the parameter is set to the latest stored value.

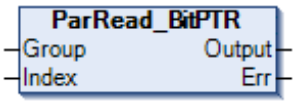
Err returns an error code if there is an error during the write operation, for example, the parameter is not found or it is a parameter of a wrong type. If the write operation is successful, *Err* returns a 0.

Pointer parameter read function block

ParRead_BitPTR

Summary

ParRead_BitPTR reads the pointed bit value from a bit pointer type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	BOOL	ANY	Output value
Err	WORD	ANY	Error output

Description

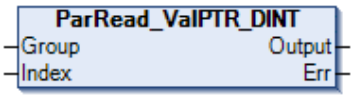
The function block reads the pointed value of a bit pointer type parameter. *Group* and *Index* define the pointed parameter to be read. The pointed value of the parameter is returned from Output.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_ValPTR_DINT

Summary

ParRead_ValPTR_DINT reads a pointed DINT/INT value from a value pointer type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	DINT	ANY	Output value
Err	INT	ANY	Error output

Description

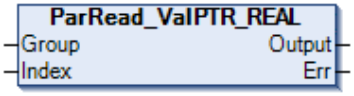
The function block reads the pointed value of a DINT or INT pointer type parameter. *Group* and *Index* define the pointed parameter to be read. The pointed value of the parameter is returned from Output. The type of Output is DINT even if the parameter type is INT.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, E*rr* returns a 0.

ParRead_ValPTR_REAL

Summary

ParRead_ValPTR_REAL reads a pointed REAL value from a value pointer type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	REAL	ANY	Output value
Err	INT	ANY	Error output

Description

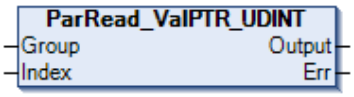
The function block reads the pointed value of a REAL pointer type parameter. *Group* and *Index* define the pointed parameter to be read. The pointed value of the parameter is returned from Output.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

ParRead_ValPTR_UDINT

Summary

ParRead_ValPTR_UDINT reads a pointed UDINT/UINT value from a value pointer type parameter.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Output	UDINT	ANY	Output value
Err	INT	ANY	Error output

Description

The function block reads the pointed value of a UDINT or UINT pointer type parameter. *Group* and *Index* define the pointed parameter to be read. The pointed value of the parameter is returned from Output. The type of Output is UDINT even if the parameter type is UINT.

Err returns an error code if there is an error during the read operation, for example, the parameter is not found or it is a parameter of a wrong type. If the read operation is successful, *Err* returns a 0.

GetPtrParConf

Summary

GetPtrParConf shows the source parameter settings. Source parameter must be value pointer, bit pointer or formatted number (parameterIndexFB).

Connections

Inputs

Name	Type	Value	Description
S_Group	INT	ANY	Parameter group
S_Index	INT	ANY	Parameter index

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output 0 = OK 3 = Invalid mapping index
S_Type	INT	0-6	Source parameter type: 0 = Unsigned 16-bit integer 1 = Signed 16-bit integer 2 = Unsigned 32-bit integer 3 = Signed 32-bit integer 4 = 32-bit Value pointer 5 = 32-bit Floating pointer 6 = 32-bit Bit pointer
T_Group	INT	ANY	Destination parameter group.
T_Index	INT	ANY	Destination parameter index.
BitNro	INT	0-31	Bit number, when bit mapping is used.
ListIdx	INT	0-N	Index of list, when list is used.
Format	INT	0-5	Shows the selected type of mapping (external interface). Not available (0) u32MAPFLAG_I16 (1) u32MAPFLAG_U16 (2) u32MAPFLAG_FLOAT (3) u32MAPFLAG_I32 (4) u32MAPFLAG_U32 (5)

Description

The block shows the source parameter settings.

If the source parameter type is formatted number/parameterIndexFB, then the parameter supports additional selection dialog (Other) in tools (selection list), where external interface selection can be changed. Format pin is showing the selection.

Based on this information (16bit/32bit/Float), original destination parameter(s) can be referenced by other blocks.

This is useful for example in cases, where the same destination parameter has different scaling factors, depending on mapped data type (16 bit or 32 bit).

Note that this selection is not affecting into interface, which is used by source parameter and in case the source parameter is application parameter with option formatted number/parameterIndexFB, it cannot be directly used by any other blocks.

When value pointer type source parameter is mapped into some destination parameter, *T_Group* and *T_Index* shows the destination parameter.

If the source parameter points into application variable, it cannot be mapped. All the other outputs are 0.

If the source parameter (parameterIndexFB) is supporting external interface settings with Set pointer parameter/other, then the Format shows the selected external interface.

If the source parameter is mapped into list, then *T_Group*, *T_Index* shows the parameter, which corresponds the list member. *BitNro* shows selected bit, and *ListIdx* shows the selected list index.

If the list member represent constant value, then *T_Group* = 0. *T_index* shows either 1 (list member =TRUE) or 0 (list member=FALSE) value.

If the source parameter is mapped into bit (*BitPtr*), then *BitNro* shows selected bit number. *T_Group*, *T_Index* indicates the destination parameter.

If the source parameter is mapped into formatted number with display format parameterIndexFB, then *S_Type* is *NUMTYPE_u32* (2) and Format shows the selected external interface.

Avoid to put this block into the fast cycle and keep the amount of blocks (instances) to minimum.

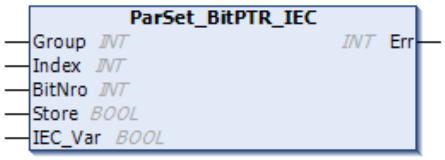
Set pointer parameter to IEC variable function blocks

Note: The old applications which are using these blocks of the earlier system library version (1.9.0.x) must be updated to the new library version (1.9.1.0.) Otherwise the application loading fault xxx occurs (aux code : 0x800A). You can also notice that the old *Par_set_ValPtr_IEC_xx* are storing the value by default and new block must have store input TRUE to have equal function. However it is not recommend to use Store option if the value is changed repeatedly.

■ ParSet_BitPTR_IEC

Summary

ParSet_BitPTR_IEC sets a bit pointer parameter to point to a bit type IEC variable.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
BitNro	INT	0	Bit setting is not supported.
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.
IEC_Var	BOOL	T/F	IEC variable

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block sets a bit pointer type parameter to point to an IEC variable of the Boolean type, that is, the IEC variable overwrites the value of the bit pointer. The parameter to point must be bit pointer type. *Group* and *Index* define the parameter. The *BitNro* input must be set to zero since (at least in this library version) the type of *IEC_Var* must be Boolean and bit pointer type parameter. Therefore the bit number cannot be chosen. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes the setting. The *IEC_Var* input is the IEC variable to be pointed.

Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, then the *Err* returns a 0.

■ ParSet_ValPTR_IEC_DINT

Summary

ParSet_ValPTR_IEC_DINT sets a value pointer parameter to point to a DINT type IEC variable.

Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.
IEC_Var	DINT	ANY	IEC variable

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block sets a value pointer type parameter to point an IEC variable of the DINT type, that is, the IEC variable value overwrites the value of the value pointer. The parameter to point must be a value pointer to the DINT or INT type. *Group* and *Index* define the parameter. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes this setting. The *IEC_Var* input is the IEC variable to be pointed.

Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, *Err* returns a 0.

ParSet_ValPTR_IEC_REAL

Summary

ParSet_ValPTR_IEC_REAL sets a value pointer parameter to point to a REAL type IEC variable.

Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.
IEC_Var	REAL	ANY	IEC variable

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

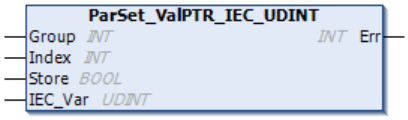
The function block sets a value pointer type parameter to point to an IEC variable of the REAL type, that is, the IEC variable value overwrites the value of the value pointer. The parameter to point must be a value pointer to the REAL type. *Group* and *Index* define the parameter. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes this setting. The *IEC_Var* input is the IEC variable to be pointed.

Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, *Err* returns a 0.

ParSet_ValPTR_IEC_UDINT

Summary

ParSet_ValPTR_IEC_UDINT sets a value pointer parameter to point to a UDINT type IEC variable.



Connections

Inputs

Name	Type	Value	Description
Group	INT	ANY	Parameter group
Index	INT	ANY	Parameter index
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.
IEC_Var	UDINT	ANY	IEC variable

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block sets a value pointer type parameter to point an IEC variable of the UDINT type, that is, the IEC variable value overwrites the value of the value pointer. The parameter to point must be a value pointer to the UDINT or UINT type. *Group* and *Index* define the parameter. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes this setting. The *IEC_Var* input is the IEC variable to be pointed.

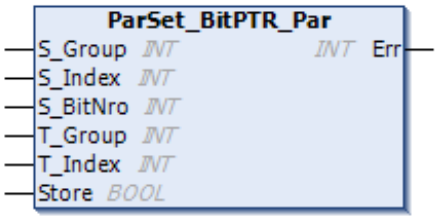
Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, *Err* returns a 0.

Set pointer parameter to parameter function blocks

■ ParSet_BitPTR_Par

Summary

ParSet_BitPTR_Par sets a bit pointer parameter to point to a bit of a packed Boolean parameter.



Connections

Inputs

Name	Type	Value	Description
S_Group	INT	ANY	Source parameter group
S_Index	INT	ANY	Source parameter index
S_BitNro	INT	ANY	Source bit number
T_Group	INT	ANY	Target parameter group
T_Index	INT	ANY	Target parameter index
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block sets a bit pointer parameter to point to a bit of a packed Boolean type parameter. *S_Group* and *S_Index* define the parameter to be pointed (the source) and *S_BitNro* defines the number of the bit. *T_Group* and *T_Index* define the pointer parameter (the target) which points to the source parameter. The target parameter must be a Bit Pointer type and the source parameter must be a packed Boolean type. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes this setting.

Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, *Err* returns a 0.

ParSet_ValPTR_Par

Summary

ParSet_ValPTR_Par sets a value pointer parameter to point to a value parameter.

Connections

Inputs

Name	Type	Value	Description
S_Group	INT	ANY	Source parameter group
S_Index	INT	ANY	Source parameter index
T_Group	INT	ANY	Target parameter group
T_Index	INT	ANY	Target parameter index
Store	BOOL	T/F	New value is stored to permanent memory of the drive. Default is FALSE, but no storing.

Outputs

Name	Type	Value	Description
Err	INT	ANY	Error output

Description

The function block sets a value pointer parameter to point to a value parameter. *S_Group* and *S_Index* define the parameter to be pointed (the source). *T_Group* and *T_Index* define the pointer parameter (the target) which points to the source parameter. The target parameter must be a pointer parameter of the same type as the source parameter which must be a value parameter. The Store pin is used to save the pointer setting to the drive permanent memory. During next power up, the drive memorizes this setting.

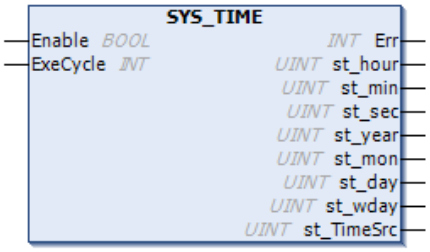
Err returns an error code if there is an error during the set operation, for example, the parameter is not found or it is a parameter of a wrong type. If the set operation is successful, *Err* returns a 0.

System time function blocks

SYS_TIME

Summary

SYS_TIME block returns to the previously set system date, time (broken time) and source.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enable/disable block execution (level sensitive).
ExeCycle	INT	ANY	Execution cycle of this clock. Not used so far, leave unconnected.

Outputs

Name	Type	Value	Description
Err	INT	ANY	Enable = 0, Err = 1. Otherwise the value must be 0.
st_hour, …, st_day	UINT	ANY	Calendar time.
st_wday	UINT	1...7	Day of the week. 1 = Monday, 7 = Sunday
st_TimeSrc	UINT	0...13	Source where the time has been set last.

Description

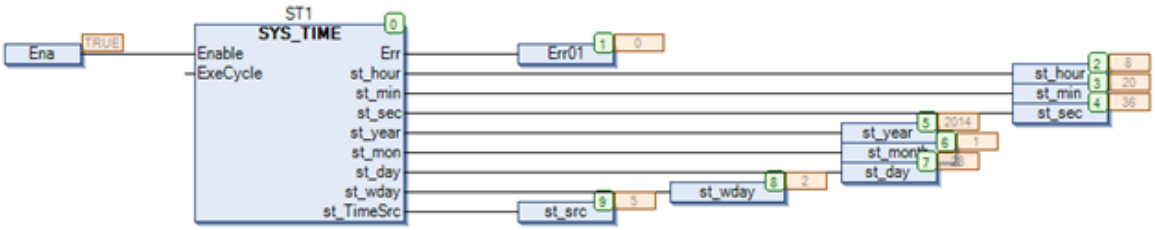
The function block use the time as the parameters 96.24...96.26, separated for easy use. To reduce the application overload (quota), set *Enable* pin to TRUE to get the time, otherwise set it to FALSE. You can put the block into slowest possible execution cycle like 500 ms. Do not use several instances of this block, only one per application.

The possible time sources given by output *st_TimeSrc* are:

Value	Description
0	Drive is maintaining its own Drive On Time.
1	User’s panel example, ACS-AP-I or DCP tool.
2	F-type of fieldbus module A.
3	D2D communication master.
4	ACS800M automation PLC via CI858, Modulebus.
5	System real time clock (RTC).

Value	Description
6	F-type of fieldbus module B.
7	Embedded fieldbus.
8	Ethernet port in BCU (ABB SAP).
9	-
10	Drive composer tool in Ethernet link (ABB SAP).
11	INU-ISU link.
12	Master follower link.
13	Time via date and time parameters.

The figure below shows an example of SYS_TIME function block, where the drive time is set by system RTC (real time clock).



SYS_TIME_UDINT

Summary

SYS_TIME_UDINT returns to the previously set system date and time (raw time) in native format (1s units) and source.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enable/disable block execution (level sensitive).
Start year	UINT	ANY	Not used so far, leave unconnected.

Outputs

Name	Type	Value	Description
Err	INT	ANY	Enable = 0, Err = 1. Otherwise the value must be 0.
TimeUDINT	UINT	ANY	Raw time (native time) in 1s units.
TimeSrc	UINT	0...13	Source where the time has been set last.

Description

To reduce the application overload (quota), set *Enable* pin to TRUE to get the time, otherwise set it to FALSE. You can put the block into slowest possible execution cycle like 500 ms (exp.Task_3). Do not use several instances of this block.

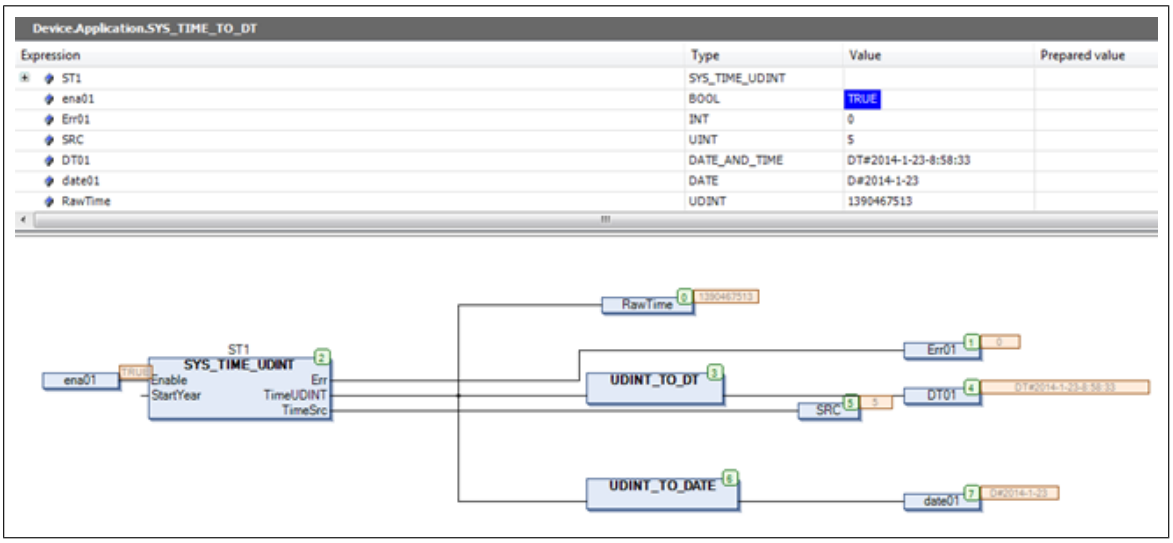
The block is intended to use together with (raw) time manipulative blocks like *UDINT_TO_DT* and *UDINT_TO_DATE*, which will convert (raw) time into IEC standard formats.

The possible time sources given by output *TimeSrc* are:

Value	Description
0	Drive is maintaining its own Drive On Time.
1	User’s panel example, ACS-AP-I or DCP tool.
2	F-type of fieldbus module A.
3	D2D communication master.
4	ACS800M automation PLC via CI858, Modulebus.
5	System real time clock (RTC).
6	F-type of fieldbus module B.
7	Embedded fieldbus.
8	Ethernet port in BCU (ABB SAP).
9	-

Value	Description
10	Drive composer tool in Ethernet link (ABB SAP).
11	INU-ISU link.
12	Master follower link.
13	Time via date and time parameters.

The below figure shows an example of *SYS_TIME_UDINT* function block, where the time is set by target RTC (real time clock).



Task time level function block

UsedTimeLevel

Summary

UsedTimeLevel block shows the time level (ms) of the program (task execution cycle) where the function block is located.

Connections

Inputs

Name	Type	Value	Description
NONE			

Outputs

Name	Type	Value	Description
Output	INT	ANY	Used time level in ms

Description

The function block shows the time level of the program (task cycle) in which the particular function block is located. Output gives the time level in milliseconds.

Error codes

The following list gives the most common error codes related to the function blocks of the ABB drives system library. The error codes are received from the *Err* output and they indicate if there is an error during the performance of the function block.

Error code	Error code number	Description
e_success	0 (hex 0)	Success, no error.
e_WriteProtected	4 (hex 4)	The parameter is write-protected.
e_Hidden	5 (hex 5)	The parameter is hidden.
e_illegalOperation	6 (hex 6)	Illegal operation, for example, the parameter type is incorrect.
e_lowLimit	9 (hex 9)	Parameter minimum value is exceeded.
e_highLimit	10 (hex A)	Parameter maximum value is exceeded.
e_noValueInList	11 (hex B)	No value in the list.
e_parNotFound	13 (hex D)	The parameter is not found.
e_OutsideIndexArea	774 (hex 306)	Outside index area.
e_OverLappingGroup	775 (hex 307)	Overlapping group.
e_UffError	777 (hex 309)	UFF error.

13

ABB D2D function blocks

Contents of this chapter

This chapter contains detailed information of the drive to drive (D2D) communication function blocks of the ABB drives D2DComm library *AY2LB_D2DComm_ACS880_V3_5*.

Introduction to ABB D2D function blocks

The ABB D2D function blocks are intended to use with the ACS880 drives. It provides drive to drive communication and drive to drive configuration function blocks for application programming in the Drive Application Builder environment. The description of the features in this document is based on the ABB drives D2D communication library version 1.9.0.2.

Note: In the Drive Composer Pro system information, make sure that the drive is installed with the corresponding system library. In System info, the D2DComm library version is located under the **Products**/ **More** view. The D2DComm library versions must be same in the drive and the application program project.

D2D communication library

Function block name	Description
Data read/write	
DS_ReadLocal	Reads data from the local dataset.
DS_WriteLocal	Writes data to local dataset.
Drive to drive communication	
D2D_TRA	Transmits data to a remote drive.
D2D_REC	Receives data from the remote drive.
D2D_TRA_REC	Transmits and receives data from the remote drive.
D2D_TRA_MC	Transmits multicast messages to group of drives.
Drive to drive configuration	
D2D_Conf	Configures token management on master drive.
D2D_Conf_Token	Configures the node related transmission cycle of token on master drive.
D2D_Master_State	Returns status of master drive connected with D2D link, except its own status.

D2D block error codes

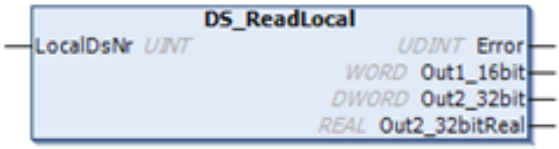
Bit	Value	Description
0	D2D_MODE_ERR	D2D is not active or message type is not supported in current D2D mode (Master/ Follower).
1	LOCAL_DS_ERR	Local dataset number out of range (1…255).
2	TARGET_NODE_ERR	Target node out of range 1...62.
3	REMOTE_DS_ERR	Remote dataset number out of range (128…255).
4	MSG_TYPE_ERR	Unsupported message type (value out of range 0...5).
5	TOO_SHORT_CYCLE	Communication overload (short token cycle).
6	INVALID_INPUT_VAL	Input value out of range (Target node and/or cycle time).
7	GENERAL_D2D_ERR	Some unspecified error situation in D2D driver.
8	RESPONSE_ERR	Syntax error in the received response.
9	TRA_PENDING	Message not sent.
10	REC_PENDING	Response not received.
11	REC_TIMEOUT	No response received.
12	REC_ERROR	Frame error in reception.
13	REJECTED	Message has been removed from the transmit buffer.
14	BUFFER_FULL	Transmit buffer is full.
15	D2D_NOT_SUPPORTED	Target is not supporting D2D.

Data read/write blocks

DS_ReadLocal

Summary

DS_ReadLocal block reads the dataset value from the local dataset table. The 48-bit dataset composes of 16-bit and 32-bit parts. The 32-bit part is available both in DWORD or REAL data formats in the function block output. The input is a pointer to the actual data.



The dataset composes of three words in the output:

- 16-bit (WORD)
- 32-bit (DWORD or REAL)

Connections

Inputs

Name	Type	Value	Description
LocalDsNr	UINT	1...255	Local dataset number

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output
Out1_16bit	WORD	ANY	16-bit part of the dataset in WORD format
Out2_32bit	DWORD	ANY	32-bit part of the dataset as DWORD format
Out2_32bitReal	REAL	ANY	32-bit part of the dataset as REAL format

Description

The function block reads the local dataset value from the local dataset table. *LocalDsNr* defines the local dataset number.

- Output Out1_16bit returns the first 16-bit of dataset as WORD data.
- Output Out2_32bit returns 32-bit part of dataset as DWORD data.
- Output Out2_32bitReal returns 32-bit part of dataset as REAL data.

Error returns an error code if there is an error during the read operation, for example, the dataset is not found or if the dataset is beyond the dataset number range of 1…255. If the read operation is successful, *Error* returns a 0.

DS_WriteLocal

Summary

DS_WriteLocal block writes data to local dataset. The 48-bit dataset composes of 16-bit and 32-bit parts. Inputs are pointers to actual data.

Connections

Inputs

Name	Type	Value	Description
LocalDsNr	UINT	128...255	Local dataset number
pDataIn1_16bit	WORD POINTER	-	Pointer to 16-bit value
pDataIn2_32bit	DWORD POINTER	-	Pointer to 32-bit data (REAL, DWORD)

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output

Description

The DS_WriteLocal function writes data to the local dataset. *LocalDsNr* defines the local dataset number from 128…255. The input data of 16-bit and 32-bit is connected to the pointer inputs *pDataIn1_16bit* and *pDataIn2_32bit* respectively using the ADR operand.

Note: The data set numbers 128…255 are reserved for application programming. However, you can set the data set numbers 1…127. There is risk of conflict with firmware dataset.

Error returns an error code if there is an error during the write operation, for example, the dataset is not found or if the dataset is beyond the dataset number range of 128…255. If the write operation is successful, *Error* returns a 0.

D2D communication blocks

General

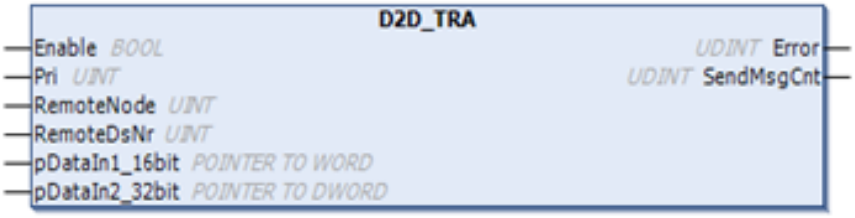
The D2D_TRA, D2D_REC and D2D_TRA_REC blocks can be used only in a master drive. These blocks can work independently without token configuration. The D2D_TRA_MC block can be used in both master and follower drives. When D2D_TRA_MC block is used in a follower drive, the token send configuration must be done using D2D_Conf_Token and D2D_Conf blocks.

The D2D_Master_State block can be used without token configuration in both the master and follower drives as well as the local dataset blocks DS_ReadLocal and DS_WriteLocal.

D2D_TRA

Summary

D2D_TRA block sends data from a Master drive to a remote Follower drive. The 48-bit data composes of 16-bit and 32-bit parts. The input data is directly given to the function block inputs, so local datasets are not required.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables sending data.
Pri	UINT	1/2	Defines the priority of sending data; Standard (1) or Low priority (2).
RemoteNode	UINT	1...62	Defines the remote drive node address.
RemoteDsNr	UINT	128...255	Defines the remote drive dataset number.
pDataIn1_16bit	WORD POINTER	-	Pointer to 16-bit value.
pDataIn2_32bit	DWORD POINTER	-	Pointer to 32-bit data (REAL, DWORD).

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.
SendMsgCnt	UDINT	ANY	Counts successfully transmitted messages.

Description

The D2D_TRA function sends application variables data from the master drive to a remote follower drive. The *Enable* input enables or disables sending data. At the rising edge of *Enable* input *Pri*, *RemoteNode* and *RemoteDsNr* are used. The input *Pri* defines the priority of data transmission.

- Standard (1): The priority is set to Standard if fast response (2 ms) is required. However, maximum of 2 blocks can be executed in the same cycle.
- Low priority (2): The priority is set to Low priority if slow response is required. It is possible to execute up to 64 blocks in the same cycle.
    - 10 ms cycle time - 10 blocks are executed
    - 100 ms cycle time - 64 blocks are executed

The inputs *RemoteNode* and *RemoteDsNr* define the remote drive node address and dataset number respectively, where the data is sent and stored. The input data of 16-bit and 32-bit is connected to the pointer inputs *pDataIn1_16bit* and *pDataIn2_32bit* respectively using ADR operand.

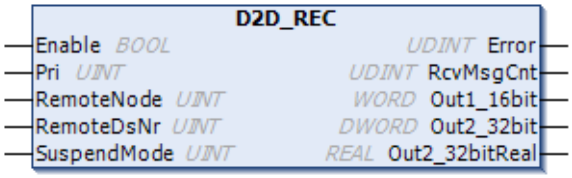
Error blocks input values and operation status if there is an error while sending data. If data is sent successfully, *Error* returns a 0. The *SendMsgCount* tracks the number of successfully sent messages.

For details of how data is sent in WORD and REAL data format to remote drive, see section *Example 1: D2D_TRA / D2D_REC blocks*.

D2D_REC

Summary

D2D_REC block enables the master drive to receive data from a remote follower drive. The block receives one 48-bit dataset from the follower dataset table. The response is available at the output signals in 16-bit and 32-bit parts. An additional 32-bit data is available in REAL format as own output.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables receiving data.
Pri	UINT	1/2	Defines the priority of receiving data; Standard (1) or Low priority (2).
RemoteNode	UINT	1...62	Defines the remote drive node address.
RemoteDsNr	UINT	128...255	Defines the remote drive dataset number.
SuspendMode	UINT	0/1	Defines the behaviour of the application task whether the D2D message is sent. 0 = message not sent 1 = message sent

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.
RcvMsgCnt	UDINT	ANY	Counts successfully received messages
Out1_16bit	WORD	ANY	16-bit dataset output value
Out2_32bit	DWORD	ANY	32-bit dataset output value
Out2_32bitReal	REAL	ANY	32-bit dataset output value in Real format.

Description

The D2D_REC block receives data from the remote drive. The *Enable* input enables or disables receiving data. At the rising edge of *Enable* input, the inputs *Pri*, *RemoteNode*, *RemoteDsNr* and *SuspendMode* are used. The input *Pri* defines the priority of receiving data.

- Standard (1): The priority is set to Standard if fast response (2 ms) is required. However, maximum of 2 blocks can be executed in the same cycle.

- Low priority (2): The priority is set to Low priority if slow response is required. It is possible to execute up to 64 blocks in the same cycle.
  - 10 ms cycle time - 10 blocks are executed
  - 100 ms cycle time - 64 blocks are executed

The inputs *RemoteNode* and *RemoteDsNr* define the remote drive node address and dataset number respectively. The remote node number is set using parameter 60.02 in the ACS880 Primary Control Program. The input *SuspendMode* defines the behavior of the application task whether the intended message is sent.

0 = continues actual application task execution

1 = indicates that actual application task execution is pending to send messages and to receive response of messages sent.

Error blocks input values and operation status if there is an error while receiving data. If receiving data is successful, *Error* returns a 0. The *RcvMsgCount* tracks the number of successfully received messages.

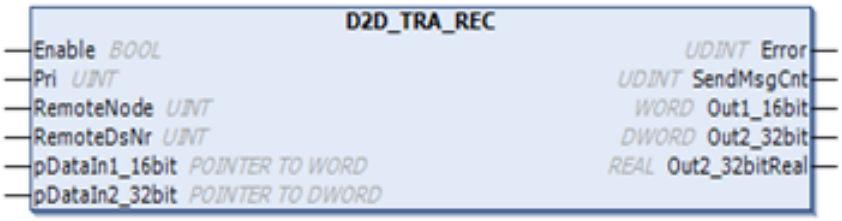
The 16-bit and 32-bit data at the output returns from *Out1_16bit* and *Out2_32bit* respectively. The 32-bit data of real data format returns from *Out2_32bitReal*.

For details of receiving data to master drive, see section *Example 1: D2D_TRA / D2D_REC blocks*.

D2D_TRA_REC

Summary

D2D_TRA_REC block enables the master drive to send and receive data from the remote drive. The 16-bit and 32-bit parts of the dataset are defined in the corresponding pointer type inputs. The response is available at the output signal in 16-bit and 32-bit parts. An additional 32-bit data is available in REAL format as own output.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables receiving data.
Pri	UINT	1/2	Defines the priority of receiving data; Standard (1) or Low priority (2).
RemoteNode	UINT	1...62	Defines the remote drive node address.
RemoteDsNr	UINT	128...255	Defines the remote drive dataset number.
pDataIn1_16bit	WORD POINTER	ANY	16-bit value connecting through ADR block.
pDataIn2_32bit	DWORD POINTER	ANY	32-bit integer or real value connecting through ADR block.

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.
SendMsgCnt	UDINT	ANY	Counts successfully transmitted messages
Out1_16bit	WORD	ANY	16-bit dataset output value
Out2_32bit	DWORD	ANY	32-bit dataset output value
Out2_32bitReal	REAL	ANY	32-bit dataset output value in Real format.

Description

The D2D_TRA_REC block sends data from the master drive and receives data from the remote drive. The *Enable* input enables/disables sending or receiving data. At the rising edge of *Enable* input, the inputs *Pri*, *RemoteNode* and *RemoteDsNr* are used. The input Pri defines the priority of receiving data.

- Standard (1): The priority is set to Standard if fast response (2 ms) is required. However, maximum of 2 blocks can be executed in the same cycle.
- Low priority (2): The priority is set to Low priority if slow response is required. It is possible to execute up to 64 blocks in the same cycle.
  - 10 ms cycle time - 10 blocks are executed
  - 100 ms cycle time - 64 blocks are executed

The inputs *RemoteNode* and *RemoteDsNr* define the remote drive node address and dataset number respectively. The response data is read from the dataset number *RemoteDsNr+1* of the remote drive. The data is selected using pointer inputs *pDataIn1_16bit* and *pDataIn2_32bit*.

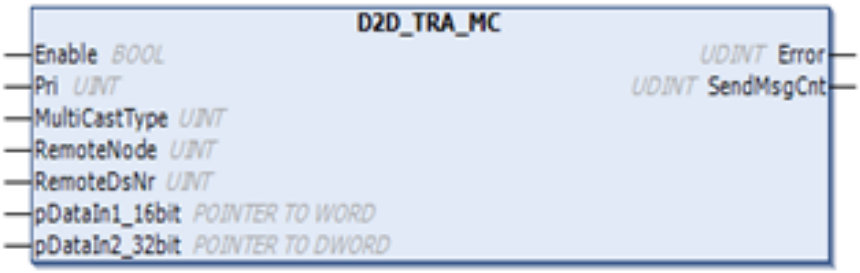
Error blocks input values and operation status if there is an error while sending or receiving data. If sending or receiving data is successful, *Error* returns a 0. The *SendMsgCount* tracks the number of successfully sent messages.

The 16-bit and 32-bit data at the output returns from *Out1_16bit* and *Out2_32bit* respectively. The additional output *Out2_32bitReal* returns 32-bit data in REAL data format.

D2D_TRA_MC

Summary

D2D_TRA_MC block enables the drive (Master or Follower) to send multicast messages to a group of drives. The block also allows sending follower to follower point to point messages.



The multicast address is defined in the *D2D_Conf block*.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables receiving data.
Pri	UINT	1/2	Defines the priority of receiving data; Standard (1) or Low priority (2).
MultiCastType	UINT	0/1	Allows sending multicast message types.
RemoteNode	UINT	1...62	Defines the remote drive node address.
RemoteDsNr	UINT	128...255	Defines the remote drive dataset number.
pDataIn1_16bit	WORD POINTER	ANY	16-bit value connecting through ADR block
pDataIn2_32bit	DWORD POINTER	ANY	32-bit integer or real value connecting through ADR block

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.
SendMsgCnt	UDINT	ANY	Counts successfully transmitted messages

Description

The D2D_TRA_MC block sends multicast messages to a group of drives. It is possible for the Master drive to receive messages from the Follower driver. For sending point to point messages or standard multicast messages, the Follower drives need token messages from the Master drive.

The *Enable* input enables/disables sending data. At the rising edge of *Enable* input the inputs *Pri*, *MultiCastType*, *RemoteNode* and *RemoteDsNr* are used.

The input Pri defines the priority of receiving data.

- Standard (1): The priority is set to Standard if fast response (2 ms) is required. However, maximum of 2 blocks can be executed in the same cycle.
- Low priority (2): The priority is set to Low priority if slower response is sufficient. Up to 64 blocks can be executed in the same cycle.
    - 10 ms cycle time - 10 blocks are executed
    - 100 ms cycle time - 64 blocks are executed

The input MultiCastType enables sending multicast messages of 3 different types:

- Follower point to point transmit (3)
- Standard Multicast (4): This message type requires all Follower/Master drives to have a corresponding multicast address equal to the RemoteNode.
- Broadcast (5): In this message type all drives in the drive to drive link receive the message including the Master drive. In this mode, the input RemoteNode must be set to 255.

The inputs *RemoteNode* and *RemoteDsNr* define the remote drive node address and dataset number respectively. The data is selected using pointer inputs *pDataIn1_16bit* and *pDataIn2_32bit*.

Error blocks input values and operation status if there is an error while sending or receiving data. If sending or receiving data is successful, *Error* returns a 0. The *SendMsgCount* tracks the number of successfully sent messages.

D2D configuration blocks

D2D_Conf

Summary

D2D_Conf block configures token management on the master drive. The D2D_Conf_Token block must be executed before the D2D_Conf block because configuration data is built based on the node data in D2D_Conf_Token block.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables configuration data in Master drive. The value FALSE stops sending token from master to follower(s).
MCastGrp	UINT	-	Defines multicast group address.
TokenTxmCycle	UINT	1000...10000	Sends the interval of token message. 0 = indicates that current configuration is removed

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.

Description

The D2D_Conf block is intended to execute only once, and for this reason, the block should be assigned to *Pre_Task*. However, the block can be assigned to any task. In cyclic tasks, the *Enable* input controls the execution, including run time configuration.

The configured data is effective on the master drive after enabling the *D2D_Conf* block. The Enable input enables/disables the configuration data on the master drive. The rising edge of *Enable* input triggers the configuration setup. The next rising edge overwrites the Enable input of *D2D_Conf_Token* block, even if it is set to FALSE.

The input *TokenTxmCycle* is the base transmission cycle of token. The node related transmission cycle is attained by multiplying this value set in the *D2D_Conf_Token* block.

Error blocks input values and operation status if there is an error in the configuration data. If the configuration is successful, *Error* returns a 0.

Master use

The master drive has a message queue to handle cyclic transmission of the token messages to follower drive. This queue can hold maximum 64 token messages. The standard multicast group of master drive (address) is defined by the input *MCastGrp*.

Follower use

In the follower drive, only the multicast group (*MCastGrp*) can be defined and the *TokenTxmCycle* is not used. The master drive transmit the token messages to follower drives. After receiving a token, the follower is able to transmit a message from the D2D message queue.

For example of token configuration, see section Example 2: Token send configuration using *D2D_Conf_Token* and *D2D_Conf* blocks.

D2D_Conf_Token

Summary

D2D_Conf_Token block configures the follower drive related token message send cycle. In the follower mode, the output Error is set.

Connections

Inputs

Name	Type	Value	Description
Enable	BOOL		Enables/disables the master drive from sending the token to follower drive.
RemoteNode	UINT	1...62	Defines the node address of the follower drive where the token is transmitted.
TxmCycMultiplier	UINT		Token send cycle. Multiplies the input *TokenTxmCycle* in block *D2D_Conf*. If the value is 0, node is removed from the configuration.

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.

Description

The D2D_Conf_Token block is used to configure the node related transmission cycle of token on master drive. This block is intended to execute only once from the Pre_Task. However, the block can be assigned to any task. In cyclic tasks, the Enable input controls the execution, including run time configuration. The settings are effective in the master only after executing the D2D_Conf block.

All node related D2D_Conf_Token blocks must be executed before D2D_Conf by setting the input *Enable* to TRUE. On run time in the Master drive, the *Enable* input enables/disables the use of follower node. However, this selection is overwritten at the next rising edge of Enable in the D2D_Conf block.

The *RemoteNode* and *TxmCycMultiplier* are set on the rising edge of Enable. The configuration is effective after the next rising edge of Enable in the block D2D_Conf. This configuration can be done on run time.

By setting the *TxmCycMultiplier* = 0, the node related token send can be removed permanently. At the next rising edge of Enable in D2D_Conf_Token and D2D_Conf blocks, the node is removed from the token configuration.

Error blocks input values and operation status. The *Error* messages are listed below:

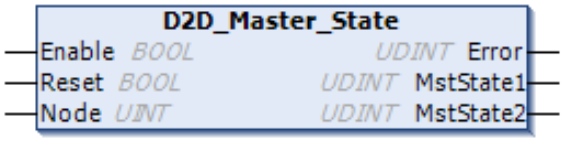
Bit	Error code	Description
0	D2D_MODE_ERR	D2D mode is not Master
5	TOO_SHORT_CYCLE	Token interval(s) are short or communication is overloaded
6	INVALID_INPUT_VAL	Input value (target node and/or cycle time) is out of range
7	GENERAL_D2D_ERR	D2D driver failed to initialize message

For example of token configuration, see section Example 2: Token send configuration using *D2D_Conf_Token* and *D2D_Conf* blocks.

D2D_Master_State

Summary

D2D_Master_State block reads bit related Master state of all the drives connected to D2D link. From the master drive, this block broadcasts the master state to other drives using node number. This block works without token management configuration.



Connections

Inputs

Name	Type	Value	Description
Enable	BOOL	T/F	Enables/disables block execution
Reset	BOOL	0/1	Resets all master state bits on rising edge
Node	UINT	1...62	Node address

Outputs

Name	Type	Value	Description
Error	UDINT	ANY	Error output.
MstState1	UDINT	0...31	Drive/node related master bits 0…31. Bit 0 == Node1
MstState2	UDINT	32...63	Drive/node related master bits 32…63.

Description

The *D2D_Master_State* block is used when there is a risk to have multiple masters in same D2D link. This enables creating systems with redundant masters. The block returns status of all Master drives connected to the D2D link, except its own state, which can be set and read using parameter *60.3* (M/F mode). As the Master drive broadcasts its state to other drives based on Node address, the panel port communication port parameter *49.1* (Node ID number) should also be using the same value.

The master drive state bits are updated when the input Reset is set FALSE. The reset function can be used whenever there is a state change from Master to Slave.

The input Node is same as parameter *60.2* (M/F node address).

Error blocks input values and operation status. In the follower drive, the output Error returns the *D2D_MODE_ERR* code to notify that the drive is not able to broadcast master state. However the block is able to read other drive states.

The output *MstState1* includes drive/node related master bits 0 to 31. If this output is set, the drive is Master.

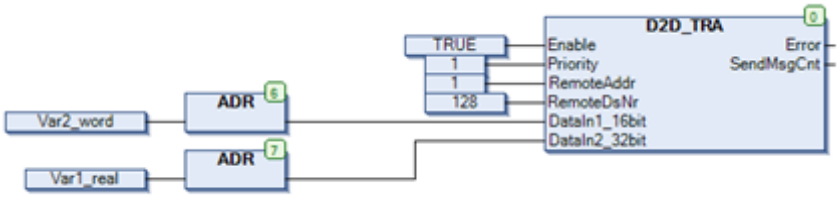
The output *MstState2* includes drive/note related master bits 32 to 63.

Examples: D2D blocks

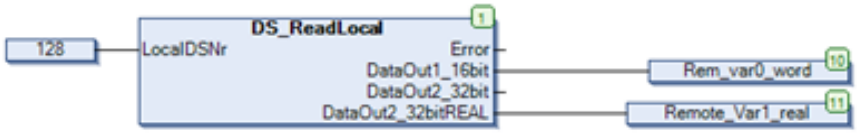
Example 1: D2D_TRA / D2D_REC blocks

The examples below describe how the *D2D_TRA* and *D2D_REC* blocks are used for sending and receiving data.

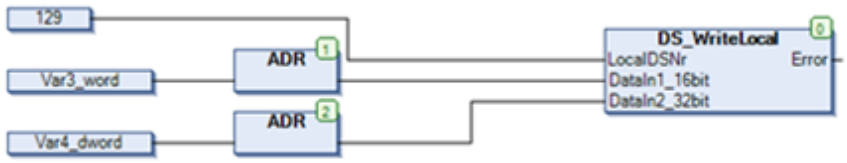
The D2D_TRA block is used to send data in WORD and REAL data format to remote drive address 1 and dataset 128.



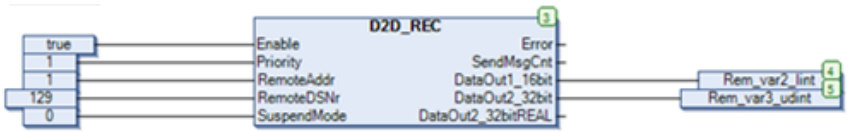
The *DS_ReadLocal* block is used to read the dataset in remote drive.



The *DS_WriteLocal* block is used to write WORD and UDINT values to remote drive dataset 129.



The *D2D_REC* block is used to receive data from the master drive.



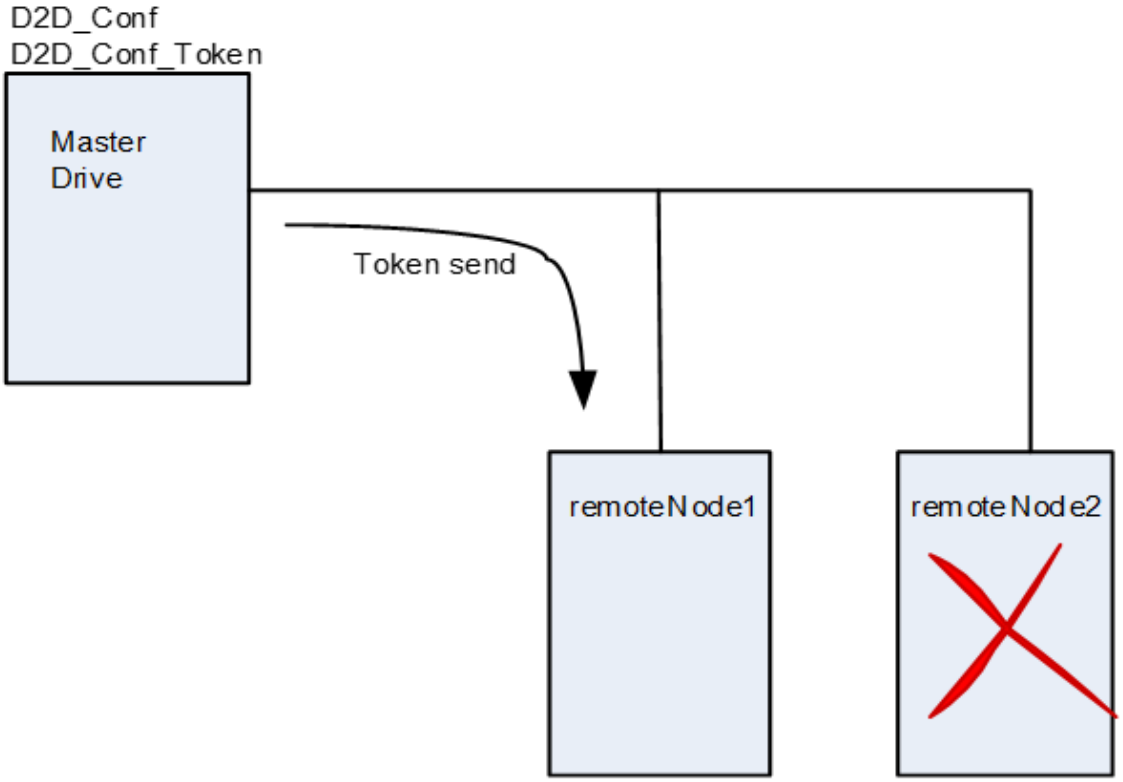
Example 2: Token send configuration blocks

The example below describes how the *D2D_Conf_Token* and *D2D_Conf* blocks are used for sending tokens.

In token send configuration, the master drive configures the token. After the follower receives a token from the master, the follower node sends follower to follower (point to point) or multicast message.

Using the *D2D_Conf_Token* block, you can add a node into the token send configuration with own instance or common instance. The example below is a common instance configuration using the *ConfToken*. When all the nodes are included, the *D2D_Conf* is executed.

In this example, a previous configuration with the following nodes existed: *remoteNode1* and *remoteNode2*. A new configuration is set that includes only *remoteNode1* for which *remoteNode2* must be removed from the existing configuration.



Each testStep represents a separate executed run cycle.

- testStep(1) - remoteNode1 is added into new configuration
- testStep(3) - remoteNode2 is removed from configuration
- testStep(4) - D2D_Conf is invoked and starts sending token to remoteNode1

```
VAR
ConfToken: D2D_Conf_Token;
ConfD2D: D2D_Conf;
VAR_END
CASE testStep OF
0: // Initialize configuration blocks
ConfToken(Enable:= FALSE);
ConfD2D(Enable:= FALSE);
testStep:= testStep + 1;
1: // Add remoteNode1 into configuration set-up (on rising edge of Enable)
ConfToken(Enable:= TRUE, TxmCycMultiplier:= 2, RemoteNode := remoteNode1);
testStep:= testStep + 1;
2: // Reset Enable pin
```

ConfToken(Enable:= FALSE);

testStep:= testStep + 1;

3: // Remove remoteNode2 from configuration set-up, by setting TxmCycMultiplier:= 0

ConfToken(Enable:= TRUE, TxmCycMultiplier:= 0, RemoteNode := remoteNode2);

testStep:= testStep + 1;

4: // Launch new D2D configuration on rising edge of Enable (start of communication with remoteNode1)

ConfD2D(Enable:= TRUE, TokenTxmCycle:= 1000);

testStep:= testStep + 1;

10: // Stop sending tokens (end of the communication)

ConfD2D(Enable:= FALSE);

testStep:= testStep + 1;

14

ABB drives standard library

Contents of this chapter

This chapter contains detailed information of the basic and special functions of the ABB drives standard library (AS1LB_Standard_ACS880_V3_5).

Overview

The ABB drives standard library is intended to use with the ACS880 drives. It provides frequently used control elements for application programming in Drive Application Builder. Unlike the standard libraries provided by 3S-Smart Software Solutions, most of the function blocks in the library use floating point numbers. This provides more flexible development environment as the programmer does not need to worry about handling wide numerical ranges and scaling.

The drive version of the library is generated from the PLC version to make sure that the code is not altered in any way. For compatibility, some functions are implemented as function blocks because the PLC does not support multiple outputs for functions. The functions do not have a state and thus require less memory. This is also why the drive version of the library has these blocks as functions (that is, there are 2 versions available in the drive version).

The input values must be within the defined limits. If the block detects that the value is out of range, then it can:

- Limit the value to the maximum or minimum value. For example, if the time constant is set to a very large value or a negative value, it is limited inside the block to make sure that it is the correct execution.
- Produce an error signal. For example, if the low limit for the output is greater than the high limit, the block cannot operate and produces an error.

The function blocks with a state has a balance reference and balance mode. This feature provides the means to force the control system to a new state. By enabling the balance

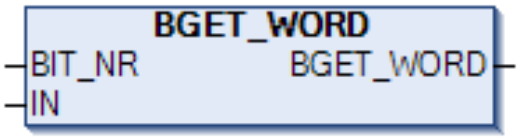
mode, the blocks operate as if the balance reference is the calculated output of the block. Internal variables are also adjusted so that once the balance mode is disabled the process continues from the balance reference value.

Basic functions

BGET

Summary

The BGET function reads one selected bit from a WORD or a DWORD (includes size check).



Connections

Inputs

Name	Type	Value	Description
BIT_NR	UINT	0...31	Bit number
IN	DWORD WORD	ANY	Data input

Outputs

Name	Type	Value	Description
BGET	BOOL	TRUE FALSE	Bit value

Function

The output (BGET) is the selected bit (*BIT_NR*) of the input word (*IN*).

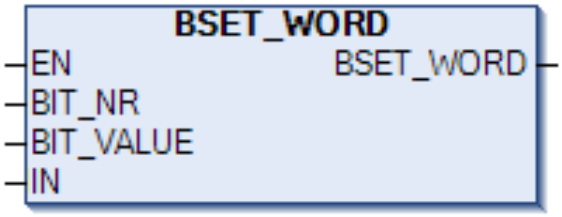
If *BIT_NR* is 0, the bit is 0. If *BIT_NR* is 31, the bit is 31.

If the bit number is not within the range of 0…31 (for DWORD) or 0…15 (for WORD), the output is 0.

BSET

Summary

The BSET function changes the state of one selected bit of a WORD or a DWORD (includes size check).



Connections

Inputs

Name	Type	Value	Description
EN	BOOL	TRUE FALSE	Enable block
BIT_NR	UINT	0...31	Bit number
BIT_VALUE	BOOL	TRUE FALSE	New value for bit
IN	DWORD WORD	ANY	Data input

Outputs

Name	Type	Value	Description
BSET	DWORD WORD	ANY	Changed word

Function

The value of a selected bit (*BIT_NR*) of the input (*IN*) is set based on the bit value input (*BIT_VALUE*).

If *BIT_NR* is 0, the bit is 0. If *BIT_NR* is 31, the bit is 31. The function must be enabled by the enable input (EN).

If the function is disabled or the bit number is not within the range of 0…31 (for DWORD) or 0…15 (for WORD), the input value is stored to the output as it is (that is, no bit setting occurs).

Example:

EN = 1, BIT_NR = 3, BIT_VALUE = 0

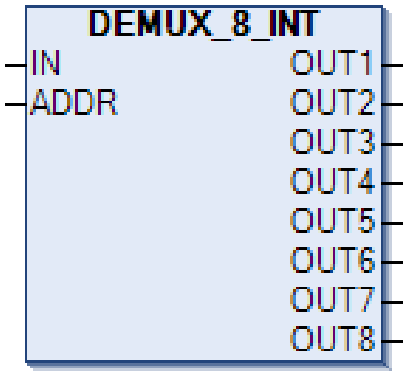
IN = 0000 0000 1111 1111

BSET = 0000 0000 1111 0111

DEMUX

Summary

The demultiplexer function block is available with 2, 4 and 8 inputs for the BOOL, DINT, INT, REAL and UDINT data types.



Connections

Inputs

Name	Type	Value	Description
IN	BOOL, DINT, INT, REAL, UDINT	ANY	Input
ADDR	UINT	1...8	Address

Outputs

Name	Type	Value	Description
OUT1…8	BOOL, DINT, INT, REAL, UDINT	ANY	Output 1...8

Function

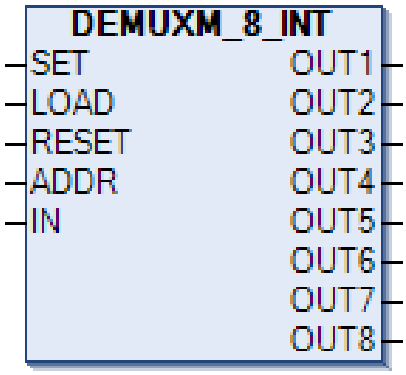
The input value (IN) is stored to the output (*OUT1…8*) selected by the address input (*ADDR*). All other outputs are set to 0.

If the address input is not from 1 to 8, all outputs are set to 0.

DEMUXM

Summary

The demultiplexer function block with an internal memory to store output values is available with 2, 4 and 8 inputs for the BOOL, DINT, INT, REAL and UDINT data types.



Connections

Inputs

Name	Type	Value	Description
SET	BOOL	TRUE, FALSE	Set
LOAD	BOOL	TRUE, FALSE	Load (Set only once)
RESET	BOOL	TRUE, FALSE	Reset
ADDR	UINT	1...8	Address
IN	BOOL, DINT, INT, REAL, UDINT	ANY	Input

Outputs

Name	Type	Value	Description
OUT1...8	BOOL, DINT, INT, REAL, UDINT	ANY	Output 1...8

Function

DEMUXM is used as a demultiplexer with the memory. It remembers the assigned value to outputs and continue to send them until changed or reset.

The input value (*IN*) is stored to the output (*OUT1…8*) selected by the address input (*ADDR*) if the load input (*LOAD*) or the set input (*SET*) is 1.

When the load input is set to 1, the input value is stored to the output only once. When the set input is set to 1, the input value is stored to the output every time the block is executed. The new set input overrides the load input.

If the address input is not from 1…8, the outputs are not affected by the input value.

If RESET = 1, all outputs are set to 0 and the block's memory is reset.

MUX

Summary

The multiplexer function is for the REAL data type. Drive Application Builder version does not support this function. The function block is available with 2, 4 and 8 inputs.

Connections

Inputs

Name	Type	Value	Description
ADDR	UINT	1...8	Address
IN1...8	REAL	ANY	Inputs 1...8

Outputs

Name	Type	Value	Description
MUX	REAL	ANY	Selected input value

Function

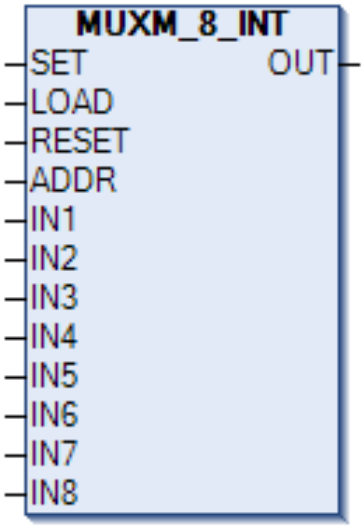
The value of an input (*IN1…8*) is selected by the address input (*ADDR*) and stored to the output (*MUX*).

If the address input is not from 1…8, the output is set to 0.

MUXM

Summary

The multiplexer function block with an internal memory to store the output is available with 2, 4 and 8 inputs for the BOOL, DINT, INT, REAL and UDINT data types.



Connections

Inputs

Name	Type	Value	Description
SET	BOOL	TRUE, FALSE	Set
LOAD	BOOL	TRUE, FALSE	Load
RESET	BOOL	TRUE, FALSE	Reset
ADDR	UINT	0...8	Address
IN1...8	BOOL, DINT, INT, REAL, UDINT	ANY	Inputs1...8

Outputs

Name	Type	Value	Description
OUT	BOOL, DINT, INT, REAL, UDINT	ANY	Output

Function

MUXM is used as a multiplexer with a memory. It remembers the assigned value of the output and continue to send them until changed or reset.

The value of an input (*IN1…8*) is selected by the address input (*ADDR*) and is stored to the output (*MUX*) if the LOAD input or the SET input is 1.

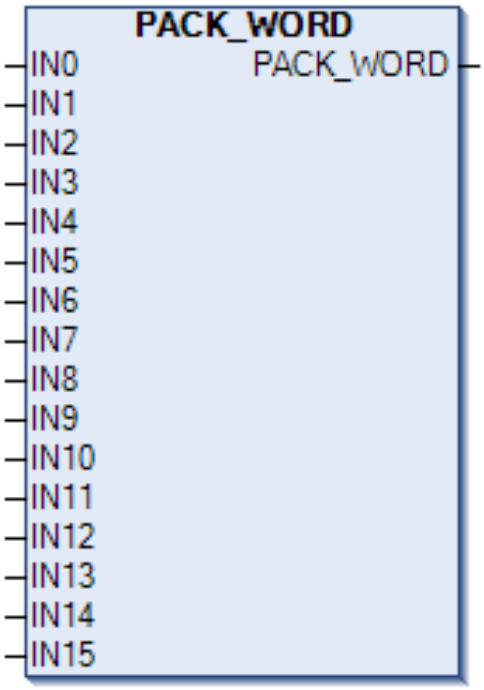
When the load input is set to 1, the input value is stored to the output only once. When the set input is set to 1, the input value is stored to the output every time the block is executed. The new set input overrides the load input.

If the address input is not from 1…8, the output is not affected by input value. If RESET = 1, then the output is set to 0 and the block's memory is reset.

PACK

Summary

The PACK function sets the BOOL inputs into a WORD or a DWORD.



Connections

Inputs

Name	Type	Value	Description
IN0...31	BOOL	TRUE, FALSE	Bits

Outputs

Name	Type	Value	Description
PACK	WORD, DWORD	ANY	Resulting pack of bits

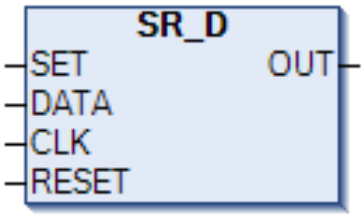
Function

The PACK function takes an input set of bits and packs it in to a word.

SR_D

Summary

The SR-D function block is an extension to a normal SR trigger with an additional memory input D trigger. The Reset signal overrides all other control signals and clears the internal block state. The Set signal forces the output to the TRUE state.



Connections

Inputs

Name	Type	Value	Description
SET	BOOL	TRUE, FALSE	Set input
DATA	BOOL	TRUE, FALSE	Data input
CLK	BOOL	TRUE, FALSE	Clock, rising edge active
RESET	BOOL	TRUE, FALSE	Reset

Outputs

Name	Type	Value	Description
OUT	BOOL	TRUE, FALSE	Output signal

Function

The SR-D block implements D trigger with the *SET*, *RESET* controls. The data is stored from D input when the clock changes from 0 to 1. The *SET* signal forces the output to the TRUE state. If R is active, the output is always FALSE. The *RESET* signal overrides all other control signals and clears the internal block state.

When the clock input (*CLK*) is set from 0 to 1, the DATA input value is stored to the output (*OUT*).

When *RESET* is set to 1, the output is set to 0.

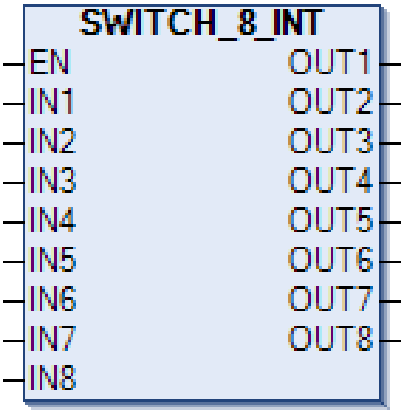
Truth table

SET	RESET	DATA	CLK	Previous output	OUT
ANY	1	Any	Any	Any	0
1	0	Any	Any	Any	1
0	0	Any	0	Q_{n-1}	Q_{n-1}
0	0	0	0 → 1	Any	0
0	0	1	0 → 1	Any	1

SWITCH

Summary

The SWITCH function block sets the output same as the input if EN equals TRUE, otherwise all outputs are 0. SWITCH is available with 2, 4 and 8 inputs and outputs for the BOOL, DINT, INT, REAL and UDINT data types.



Connections

Inputs

Name	Type	Value	Description
EN	BOOL	TRUE, FALSE	Enable
IN1...8	BOOL, DINT, INT, REAL, UDINT	ANY	Input 1...8

Outputs

Name	Type	Value	Description
OUT1...8	BOOL, DINT, INT, REAL, UDINT	ANY	Output 1...8

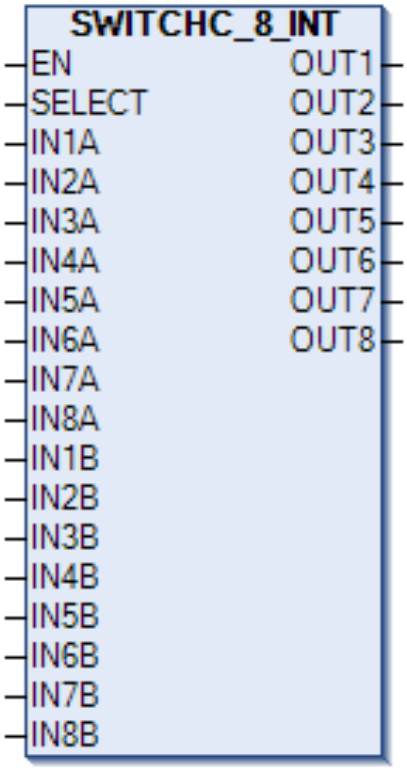
Function

The output (*OUT1…8*) is equal to the corresponding input (*IN1…8*) if the block is enabled (EN = 1). Otherwise the output is 0.

SWITCHC

Summary

The SWITCHC function block has two channels. A channel can be chosen by using the SELECT signal. If SELECT equals FALSE, channel A is active. If SELECT equals TRUE, channel B is active. If the EN signal is not active, all outputs are 0. SWITCHC is available with 2, 4 and 8 input pairs and outputs for the BOOL, DINT, INT, REAL and UDINT data types.



Connections

Inputs

Name	Type	Value	Description
EN	BOOL	TRUE, FALSE	Enable
SELECT	BOOL	True, FALSE	Select
IN1...8A	BOOL, DINT, INT, REAL, UDINT	ANY	Input A 1...8
IN1...8B	BOOL, DINT, INT, REAL, UDINT	ANY	Input B 1...8

Outputs

Name	Type	Value	Description
OUT1...8	BOOL, DINT, INT, REAL, UDINT	ANY	Output A 1...8

Function

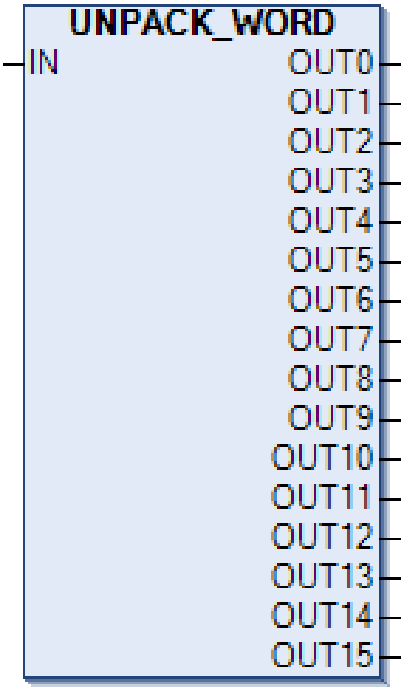
The output (*OUT1…8*) is equal to the corresponding channel A input (*IN1…8A*) if the activate input signal (*SELECT*) is 0. The output is equal to the corresponding channel B input (*IN1…8B*) if the activate input signal (*SELECT*) is 1.

If the block is disabled (EN = 0), all outputs are set to 0.

UNPACK

Summary

The UNPACK function block splits a WORD or a DWORD into a set of BOOL outputs.



Connections

Inputs

Name	Type	Value	Description
IN	WORD, DWORD	ANY	Input data

Outputs

Name	Type	Value	Description
OUT0...31	BOOL	TRUE, FALSE	Output bits

Function

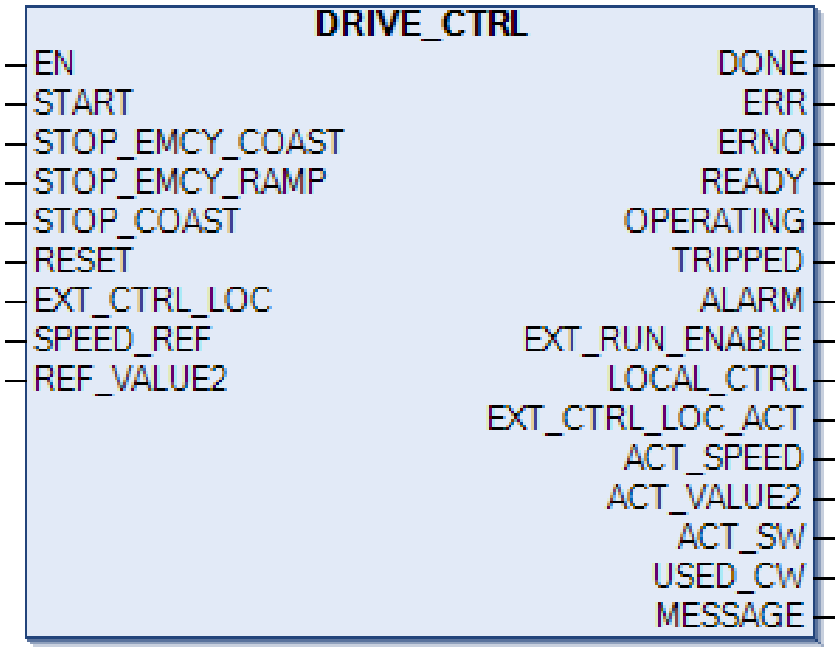
The Unpack function takes an input word and returns it as a set of bits.

Special functions

Drive control

Summary

The drive control program offers basic controls of an ACS880 drive to the application programmers. A similar function block for the PLC to control the drive exists in the PS553 library.



Connections

Inputs

Name	Type	Value	Description
EN	BOOL	TRUE, FALSE	Enable function block - TRUE. Additionally configures the drive to use the application program. See parameters *19.11*, *20.1*, *20.6*, *22.11* and *26.11*.
START	BOOL	TRUE, FALSE	TRUE = start drive FALSE = stop along currently active stop ramp. See parameter *6.2.0*.
STOP_EMCY_COAST	BOOL	TRUE, FALSE	Emergency coast stop to drive: FALSE = stop by coast TRUE = no stop See parameter *6.2.1*.
STOP_EMCY_RAMP	BOOL	TRUE, FALSE	Emergency stop to drive FALSE = stop by ramp TRUE = no stop See parameter *6.2.2*.

Name	Type	Value	Description
STOP_COAST	BOOL	TRUE, FALSE	TRUE = coast stop FALSE = normal operation See parameter *6.2.3*.
RESET	BOOL	TRUE, FALSE	Resets drive and internal parameter errors. See parameter *6.2.7*.
EXT_CTRL_LOC	BOOL	TRUE, FALSE	Selects external control location (EXT1/EXT2). See parameters *6.2.11* and *19.11*.
SPEED_REF	REAL	ANY	Speed reference value. See parameter *22.11*.
REF_VALUE2	REAL	ANY	Torque reference value. See parameter *26.11*.

Outputs

Name	Type	Value	Description
DONE	BOOL	TRUE, FALSE	Execution finished when output DONE = TRUE.
ERR	BOOL	TRUE, FALSE	Error occurred during execution when output ERR = TRUE
ERNO	ENUM	ANY	Internal error code
READY	BOOL	TRUE, FALSE	Ready to switch on See parameter *6.11.0*
OPERATING	BOOL	TRUE, FALSE	Drive is operating.
TRIPPED	BOOL	TRUE, FALSE	Drive FAULT See parameter *6.11.3*.
ALARM	BOOL	TRUE, FALSE	Drive has an alarm See parameter *6.11.7*.
EXT_RUN_ENABLE	BOOL	TRUE, FALSE	Run enable status See parameter *6.18.5*.
LOCAL_CTRL	BOOL	TRUE, FALSE	Drive control location: LOCAL See parameter *6.11.9*.
EXT_CTRL_LOC_ACT	BOOL	TRUE, FALSE	Actual external control location EXT2 selected See parameter *6.16.11*.
ACT_SPEED	REAL	ANY	Actual speed (in rpm) read from drive See parameter *1.01*.
ACT_VALUE2	REAL	ANY	Actual torque (in %) read from drive See parameter *1.10*.
ACT_SW	WORD	ANY	Main status word read from drive See parameter *6.11*.
USED_CW	WORD	ANY	Application control word See parameter *6.02*.

Name	Type	Value	Description
MESSAGE	ENUM	ANY	State of the function block

Function

The program uses drive parameters as an interface to the drive.

An application control word (*06.02*) is used to control the drive. It sets the EXT1 command (*20.01*) and EXT2 command (*20.06*) parameters to Application Program. The control word is defined in the ABB Drives control profile.

When the drive is in the operational state, the OPERATING output is set to TRUE to indicate the current state.

You can enable the program by setting EN signal to TRUE. Once active, the block sets the configuration parameters to the desired values: Parameters *19.11*, *20.01*, *20.06*, *22.11* and *26.11* are set to Application Program. The parameters are intentionally changed once (enabling the signal to TRUE) to change them manually while the program is running.

The drive status is obtained from the Main status word (*06.11*) and Status word 1 (*06.16*). The actual speed (*ACT_SPEED*) and torque (*ACT_VALUE2*) data are obtained from parameters Motor speed used (*01.01*) and Motor torque % (*01.10*).

When the program is disabled, Application control word is set to 0 once.

If the EXT1 and EXT2 parameters are not set to the correct value while the program is enabled, an error is produced.

Error codes and the ERR outputs are internal program errors and not drive fault codes. Internal parameter errors do not prevent the program from functioning.

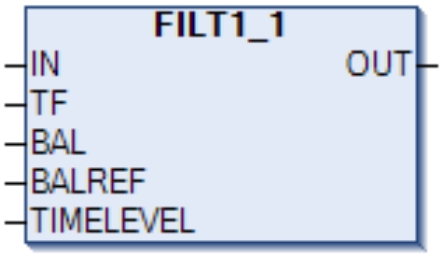
Limiting

Only one instance of drive control is allowed. This is why it is implemented as a program.

Filter

Summary

The FILT1_1 function block provides filtering of the high frequency part of the input signal. The block acts as a single-pole low pass filter for the REAL numbers. The balancing function permits the output signal to track an external reference.



Connections

Inputs

Name	Type	Value	Description
IN	REAL	ANY	Input signal for the actual value.

Name	Type	Value	Description
TF	REAL	0...ANY	Filter time constant (ms).
BAL	BOOL	TRUE, FALSE	Balance input, activates the tracking mode.
BALREF	REAL	ANY	Value for the tracking mode.
TIME-LEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms.

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Filtered actual value

Function

The function filters the input signal using the current input and previous output.

The transfer function for a single-pole filter with no pass band gain is:

$$G(s) = \frac{1}{(1 + sTF)}$$

To get the function for the output, cross-multiply the equation.

$$O(s) * (1 + sTF) = 1 * I(s)$$

Resolving the parenthesis gives:

$$O(s) + sTF * O(s) = I(s)$$

To get the equation to the time domain s has to be replaced by derivation.

$$O(t) + TF * O(t) = I(t)$$

Since this is a first order approximation function block, the derivation can be replaced by a difference.

$$O(t) + TF * \frac{O(t) - O(t-1)}{(Ts)} = I(t)$$

Where: Ts is the cycle time of the program in milliseconds (time difference between t and t-1).

The final filtering algorithm is calculated by using the following formula that is obtained by extracting O(t):

$$O(t) = \frac{1 + (TF/TS) * O(t-1)}{TF/Ts + 1}$$

If TF = 0 or negative, the output value is set to the input value.

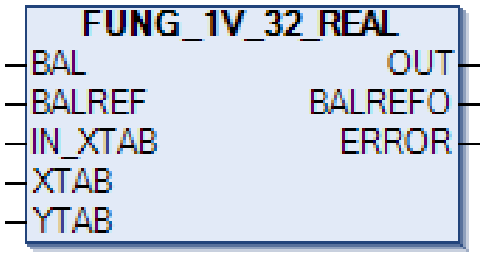
Because of the REAL data type limitation, the TF/Ts ration is limited to 8000000, to make sure that it is always possible to add 1 to the real value.

Function generator

Summary

The FUNG_1V function block is used to generate an optional function of one variable, y = f(x). The function is described by a number of coordinates. Linear interpolation is used for values between these coordinates. An array of 8, 16 or 32 coordinates can be specified.

The balancing function permits the output signal to track an external reference and gives a smooth return to the normal operation.



Connections

Inputs

Name	Type	Value	Description
BAL	BOOL	TRUE, FALSE	Input to activate the balancing mode.
BALREF	REAL	ANY	Balance reference. Input for the reference value in the balancing mode.
IN_XTAB	REAL	ANY	Input signal for the function.
XTAB	REAL [N]	ANY	Table of X coordinates for the function.
YTAB	REAL [N]	ANY	Table of Y coordinates for the function.

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Value of the function.
BALREFO	REAL	ANY	TRUE if the high limit is reached.
ERROR	BOOL	TRUE, FALSE	TRUE when the input is outside the table range or when the table contains unsorted (low to high) data for the input coordinates.

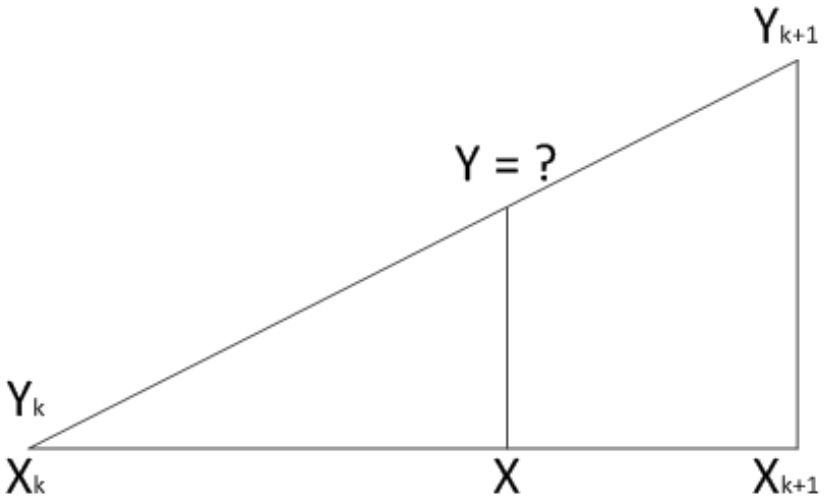
Function

The function generator FUNG_1V calculates output signal Y for a value at input X. Calculation is performed in accordance with a piece-by-piece linear function which is determined by vectors XTAB and YTAB. For each X value in XTAB, there is a corresponding Y value in YTAB. The Y value at the output is calculated by means of linear interpolation of the XTAB values, between which lies the value of input X. The values in XTAB must increase from low to high in the table.

The output of the block depends only on the current input values, in other words, it does not have any state.

Interpolation

The generated function is performed as follows:



$$Y = Y_k + \frac{(X - X_k)(Y_{k+1} - Y_k)}{(X_{k+1} - X_k)}$$

Balancing

If BAL is set to TRUE, the value at Y is set to the value of the *BALREF* input. The X value which corresponds to Y value is obtained at the *BALREFO* output. On balancing, the X value is calculated by interpolation in the same way the Y value is calculated during the normal operation. To permit balancing, the values in YTAB must increase from low to high in the table.

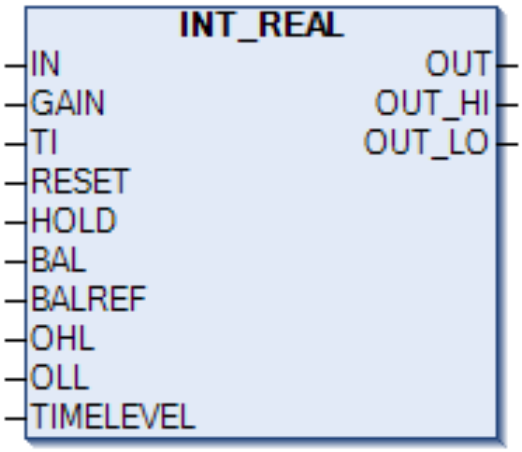
Limiting

If input signal X is outside the range defined by XTAB, the Y value is set to the highest or lowest value in YTAB. If *BALREF* is outside the YTAB value range in the *BAL* mode, the value at Y is set to the value at the *BALREF* input and *BALREFO* is set to the highest or lowest value in XTAB.

Integrator

Summary

The INT_REAL function block integrates the input. The output signal can be limited within limit values. The balancing function permits the output signal to track an external reference and gives a smooth return to the normal operation.



Connections

Inputs

Name	Type	Value	Description
IN	REAL	ANY	Input signal for the actual value
GAIN	REAL	ANY	Gain input
TI	REAL	0...ANY	Integration time (ms)
RESET	BOOL	TRUE, FALSE	Clear integrated value
HOLD	BOOL	TRUE, FALSE	Stops integration when set to TRUE
BAL	BOOL	TRUE, FALSE	Balance input, activates the tracking mode
BALREF	REAL	ANY	Value for the tracking mode
OHL	REAL	ANY	High input limit
OLL	REAL	ANY	Low input limit
TIME-LEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Output value.
OUT_HI	BOOL	TRUE, FALSE	TRUE if the high limit is reached.
OUT_LO	BOOL	TRUE, FALSE	TRUE if the low limit is reached.

Function

The INT function can be written in the time plane as:

$$O(t) = K/T_i(\int I(t)dt)$$

The main controlled property is that the output signal retains its value when the input signal *I(t)* = 0.

Clearing

The integrated value is cleared when RESET = TRUE (all internal variables are cleared).

Tracking

If *BAL* is set to TRUE, the integrator immediately goes into the tracking mode and the output value is set to the value of the *BALREF* input. If the value at *BALREF* exceeds the output signal limits, the output is set to the applicable limit value. On return to the normal operation from the tracking mode, integration continues from the tracking reference.

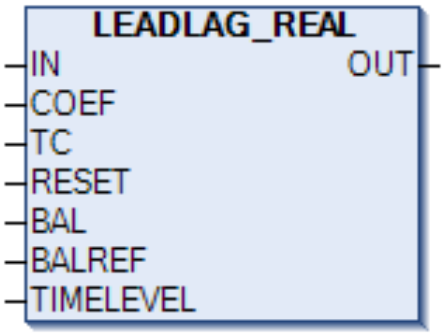
Limiting

The output value is limited between *OHL* and *OLL*. If the actual value exceeds the upper limit, the *OUT_HI* output is set to TRUE. If it falls below the lower limit, the *OUT_LO* output is set to TRUE. If the limits have incorrect values, both *OUT_HI* and *OUT_LO* are set to TRUE.

Lead lag

Summary

The LEADLAG_REAL function block is used to filter the input signal and provide a phase shifted output. This block acts as a lead/lag filter based on the COEF input value.



Connections

Inputs

Name	Type	Value	Description
IN	REAL	ANY	Input signal for the function block
COEF	REAL	ANY	Constant that determines the filter type
TC	REAL	0...ANY	Time constant (ms)
RESET	BOOL	TRUE, FALSE	Resets the function block
BAL	BOOL	TRUE, FALSE	Activates the balance mode
BALREF	REAL	ANY	Balance reference Input for the reference value in the balancing mode.
TIMELEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Output signal

Function

The transfer function for the lead/lag filter is:

$$\frac{1 + \alpha T_c s}{1 + T_c s}$$

The lead/lag filter has two input parameters TC and α (COEF):

- If α > 1, the filter acts as a lead filter.
- If α < 1, the filter acts as a lag filter.
- If α = 1, no filter is applied.

The filter algorithm is calculated using the following formula:

dn = X - B1*dnMem

Y = A0*dn + A1*dnMem

dnMem = dn

Where,

A0 = (1 + α*Tc) / (1 + Tc),

A1 = (1 - α*Tc) / (1 + Tc),

B1 = (1 - Tc) / (1 + Tc)

X is the input signal.

Y is the output signal.

The initial value of dnMem is set to zero.

Note: If *α* or *TC* input to the block is negative, the corresponding negative input is assigned to zero before the filter algorithm is calculated.

Because of the REAL data type limitation, the TC/Ts ration is limited to 8000000, to make sure that it is always possible to add 1 to the real value.

Balancing

If *BAL* is set to TRUE, the value at Y is set to the value of the *BALREF* input. The block operates normally during this time which means that the internal variable is always calculated.

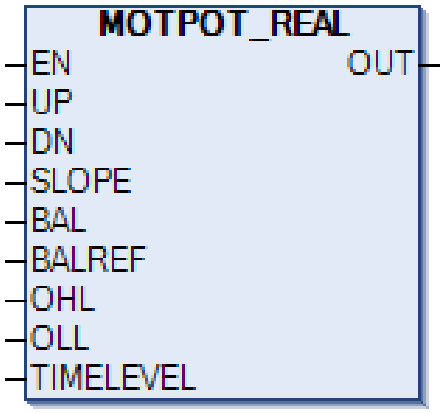
Reset

If *RESET* is set to TRUE, the internal variable *dnMem* is set to zero and input value X is returned.

Motor potentiometer

Summary

The MOTPOT_REAL (motor potentiometer) function block is used to generate the reference based on the activation of the Boolean (UP and DN) inputs. The rate of change of a reference signal is controlled by the slope time and limits. The current value is retained after a power cycle.



Connections

Inputs

Name	Type	Value	Description
EN	BOOL	TRUE, FALSE	Enables operations.
UP	BOOL	TRUE, FALSE	Enables count up.
DN	BOOL	TRUE, FALSE	Enables count down.
SLOPE	UINT	0...65535	Delay time to count from *OLL* to *OHL* and vice versa.
BAL	BOOL	TRUE, FALSE	Sets the output to *BALREF* or limit if it exceeds the limit.
BALREF	REAL	ANY	Sets the output value when the *BAL* input is active.
OHL	REAL	ANY	High input limit.
OLL	REAL	ANY	Low input limit.
TIMELEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms.

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Output value

Function

The MOTPOT functional block is used to control the rate of change of an output reference signal. Digital inputs are normally used as the *UP* and *DOWN* inputs.

The rate of change of a reference signal is controlled by the slope time parameter. If the enable pin (*EN*) is set to TRUE, the reference value rises from minimum to maximum during the slope time.

EN turns on the MOTPOT function. If *EN* is set to FALSE, the output is zero. Based on the *UP* or *DN* inputs getting activated, the output reference increases or decreases to the maximum or minimum value based on the slope. If both *UP*/*DN* inputs are activated/deactivated, the output is neither incremented nor decremented and is in a steady state.

Clearing

When *EN* is set to FALSE, the output and internal values are set to zero.

Tracking

If *BAL* is set to TRUE, the output is set to the value of the *BALREF* input. If the value at *BALREF* exceeds the output signal limits, the output is set to the applicable limit value.

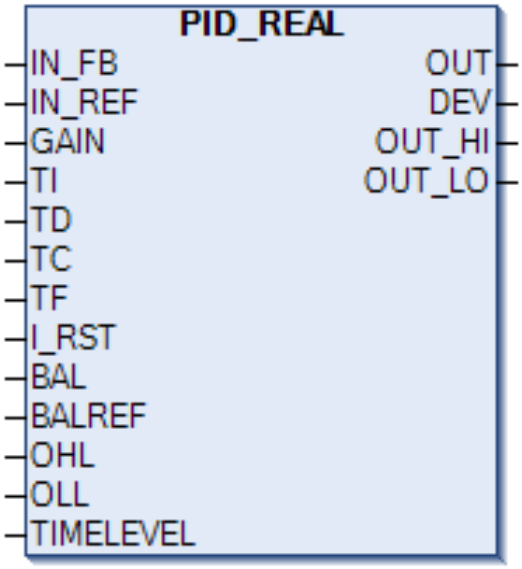
Limiting

The output value is limited between *OHL* and *OLL*. If the actual value is more than the upper limit, the output is set to the *OHL* input value. If the actual value decreases below the lower limit, the output is set to the *OLL* input value.

PID

Summary

The PID_REAL (Proportional-Integral-Derivative) element can be used as a generic PID regulator in feedback systems. The main extension of the element is that a derivative correction term with a filter is included. Another major extension is the antiwindup protection. The output signal can be limited with limit values specified at special inputs (*OHL* and *OLL*). The balancing function permits the output signal to track a gradual return to the normal operation. After any parameter change or error condition, the integral term of the correction is readjusted so that the output does not change suddenly (“bumpless transfer”).



Connections

Inputs

Name	Type	Value	Description
IN_FB	REAL	ANY	Actual input value

Name	Type	Value	Description
IN_REF	REAL	ANY	Reference input value
GAIN	REAL	ANY	Proportional gain
TI	REAL	0...ANY	Integration time (ms)
TD	REAL	0...ANY	Derivation time (ms)
TC	REAL	0...ANY	Anti-windup correction time (ms)
TF	REAL	0...ANY	Filter time (ms)
I_RST	BOOL	TRUE, FALSE	Clear integrator
BAL	BOOL	TRUE, FALSE	Balance input, activates the tracking mode.
BALREF	REAL	ANY	Value for the tracking mode
OHL	REAL	ANY	High input limit
OLL	REAL	ANY	Low input limit
TIMELEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Output signal
DEV	REAL	ANY	Deviation (*IN_FB* - *IN_REF*)
OUT_HI	BOOL	TRUE, FALSE	TRUE if the high limit is reached.
OUT_LO	BOOL	TRUE, FALSE	TRUE if the low limit is reached.

Function

The differential equation describing the PID controller before saturation/limitation that is implemented in this block is:

$$Output_{presat}(t) = Up(t) + Ui(t) + Ud(t)$$

Where

OUT_{presat} is the PID output before saturation

Up is the proportional term

Ui is the integral term with saturation correction

Ud is the derivative term

t is time.

The proportional term is:

$$Up(t) = Kp * DEV(t)$$

Where:

Kp = P is the proportional gain of the PID controller

DEV(t) is the control deviation (see below).

The integral correction term is:

$$Ui(t) = \frac{KP}{Ti} * \int DEV(\tau)d\tau + Kc * (OUT(t) - OUT_{presat}(t))$$

Where:

Kc = integral antiwindup correction gain of the PID controller

OUT(t) = saturated/limited output signal of the controller

The antiwindup correction is thus taken to be part of the integral correction term.

$$K_c * (OUT(t) - OUT_{presat}(t))$$

Windup is a phenomenon that is caused by the interaction of an error integral action and saturations. All actuators have limitations: a motor has limited speed, a valve cannot be more than fully opened or fully closed, and so on. For a control system with a wide range of operating conditions, it is possible that the control variable reaches the actuator limits. When this happens, the feedback loop is broken and the system runs as an open loop because the actuator remains at its limit independently of the process output. If a controller with the integrating action is used, the error continues to be integrated. This means that the integral term may become very large or, in other words, it “winds up”. Then it is required that the error has the opposite sign for a long period before things return to normal. The consequence is that any controller with the integral action may give large transients when the actuator saturates.

The derivative term is:

$$Ud(t) = Kp * Td * \frac{d(DEV(t))}{(dt)}$$

Where:

Td is the derivative time constant.

The differential equations above are transformed into difference equations by backward approximation.

The term is also filtered to make it resistant to high frequency noise.

$$G(s) = 1/(1 + s * TF)$$

Smooth transfer

The controller guarantees a smooth transfer in many special situations where, for example, control parameters are suddenly changed. This means that in such a bumpless cycle the output retains its previous value. This is performed by resetting the integrator term Ui to:

$$Ui(t) = OUT(t) - Up(t) - Ud(t)$$

Smooth functionality is not triggered in the first cycle by change in *Ti*, *Tc*, *Td* and *Tf*.

Gain, time constants

The proportional gain *Kp* is a direct input parameter. The integrator, derivative and antiwindup gains *Ki*, *Kd* and *Kc* must be calculated from the corresponding time constants *Ti*, *Td* and *Tc* which are input parameters. The derivative gain is:

$$Kd = Td/T$$

Where:

T is the time level (execution cycle) of the block (in milliseconds as the time constants).

The integral gain is determined from *Ti* as follows:

Ki = 0 if Ti = 0

Ki = T/Ti, if T < Ti

Ki = 1, if T ≥ Ti > 0

The anti-windup gain is determined similarly by Tc:

Kc = 0, if Tc = 0

Kc = T/Tc, if T < Tc

Kc = 1, if T ≥ 0

Thus the values of *Ki* and *Kc* are limited to the range 0 ≤ *Ki*, *Ti* ≤ 1.

If *Tc* = 0, *Kc* = 0 and anti-windup correction is disabled.

If *Ti* = 0, *Ki* = 0, the module does not update the integral term Ui, not even by the anti-windup correction. Thus the integrator term retains its original value as long as Ki remains zero.

The element stores the “current” set of gains *Kp*, *Ki*, *Kc* and *Kd* and time constants *Ti*, *Tc* and *Td*, which it uses for calculating the control output(s).

Filtering

The derivative is filtered using a single-pole low pass filter. The following algorithm is used to calculate the filtered value:

$$y(t) = \frac{Kd * (Up(t) - Up(t-1)) + \frac{Tf}{T} * y(t-1)}{1 + \frac{Tf}{T}}$$

Where,

T is the time level (execution time) of the block (in milliseconds as the time constants).

If the filter time constant is left unassigned, it defaults to 0 which means that the derivative is calculated without filtering. The time constant is limited to 8000000*time level to avoid underflow.

Tracking

If *BAL* is set to TRUE, the regulator goes into the tracking mode and the output follows the value at *BALREF*. If the value at *BALREF* exceeds the output signal limits (*OLL* and *OHL*), the output is set to the applicable limit value. The return from the tracking state is bumpless.

Limitation function

The limitation function limits the output signal to the value range from *OLL* to *OHL*. If the presaturated output exceeds *OHL*, *OUT* is set to *OHL* and *OUT_HI* is set to TRUE. If the pre-saturated output decreases below *OLL*, *OUT* is set to *OLL* and *OUT_LO* is set to TRUE. Bumpless return from limitation is requested if and only if the anti-windup correction is not in use, that is,. *Ki* = 0 or *Kc* = 0.

IF *OLL* < *OHL*, both *OUT_HI* and *OUT_LO* are set to TRUE and OUT retains the value that it had in the execution cycle before the error occurred. After the error, the return to the normal operation is smooth.

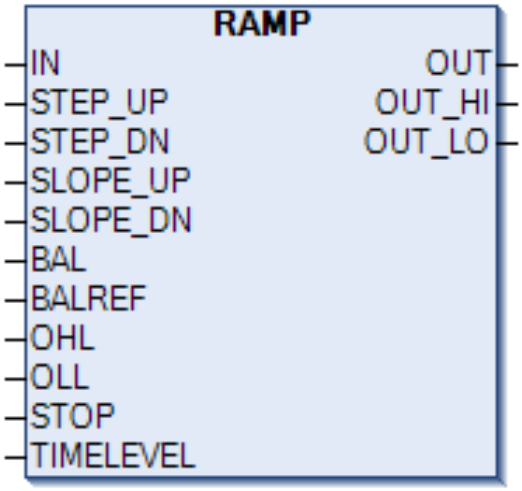
Limiting

The output value is limited between *OHL* and *OLL*. If the actual value is more than the upper limit, *OUT_HI* is set to TRUE. If the actual value decreases below the lower limit, *OUT_LO* is set to TRUE.

Ramp

Summary

The RAMP is used to limit the rate of change of a signal. The output signal can be limited with limit values specified at special inputs. The balancing function permits the output signal to track an external reference.



Connections

Inputs

Name	Type	Value	Description
IN	REAL	ANY	Input signal for the actual value
STEP_UP	REAL	0...ANY	The greatest allowed positive STEP change
STEP_DN	REAL	0...ANY	The greatest allowed negative STEP change
SLOPE_UP	REAL	0...ANY	Positive ramp for the output
SLOPE_DN	REAL	0...ANY	Negative ramp for the output
BAL	BOOL	TRUE, FALSE	Balance input, activates the tracking mode.
BALREF	REAL	ANY	Balance reference Input for the reference value in the tracking mode
OHL	REAL	ANY	High input limit
OLL	REAL	ANY	Low input limit
STOP	BOOL	TRUE, FALSE	Holds the output (stops ramping)
TIMELEVEL	INT	1...ANY	Task interval in milliseconds, default = 10 ms

Outputs

Name	Type	Value	Description
OUT	REAL	ANY	Output value
OUT_HI	BOOL	TRUE, FALSE	TRUE if the high limit is reached
OUT_LO	BOOL	TRUE, FALSE	TRUE if the low limit is reached

Function

The main property of the RAMP element is that the output signal tracks the input signal, while the input signal is not changed more than the value specified at the step inputs. If the input signal change is more than the specified step change, the output signal is first changed by *STEP_UP* or *STEP_DN* depending on the direction of change. After the output signal is changed by *SLOPE_UP* or *SLOPE_DN* per second, until the values at the input and output are equal. This means that if *STEP_DN* = *STEP_UP* = 0, a pure ramp function, that is, SLOPE/sec is obtained at the output. The greatest step change allowed at the output is specified by the *STEP_UP* and *STEP_DN* inputs for the respective direction of change.

All parameters are specified as absolute values with the same unit as the input. Slopes specify the change in units per second. Certain constants are pre-calculated to make the execution time of the element as short as possible. The results are stored internally in the element. These constants are recalculated if the *SLOPE_UP* or *SLOPE_DN* values are changed.

Calculation of the output

If Input (t) = Output (t-1), then Output (t) = Input (t)

If Input (t) > Output (t-1), then the change of the output value is limited as follows:

- An internal auxiliary variable VPOS follows the input value with the maximum rate of change defined by *SLOPE_UP*. If the input value is greater than VPOS + STEP_UP, the output value is limited to the value VPOS +STEP_UP. If the input value is less than VPOS + STEP_UP, the output value is set to be equal to the input.

If *SLOPE_UP* = 0, the output value does not rise.

If Input (t) < Output (t-1), then the change of the Output value is limited as follows:

- An internal auxiliary variable VPOS follows the input value, with the maximum rate of change defined by *SLOPE_DN*. If the input value is less than VPOS – STEP_DN, the output value is limited to the value VPOS – STEP_DN. If the input value is greater than VPOS – STEP_DN, the output value is set to be equal to the input.

If *SLOPE_DN* = 0, the output value does not lower no matter what the value of *STEP_DN* and *IN* is.

Tracking

If BAL is set to TRUE, the ramp immediately goes into the tracking mode and the output is set to the value of *BALREF*. If the value at *BALREF* exceeds the output signal limits, the output is set to the applicable limit value. During the tracking mode VPOS = Output = *BALREF*. The return to the normal operation is done as if a unit step had occurred at the input.

Limiting

The limitation function limits the output signal to the values at the *OHL* inputs for the upper limit and *OLL* for the lower limit. If the actual value exceeds the upper limit, *OUT_HI* is set to TRUE. If it falls below the lower limit, *OUT_LO* is set to TRUE. In the limiting state VPOS(t) and OUT(t) are set to the applicable limit value.

If *OLL* < *OHL*, both *OUT_HI* and *OUT_LO* are set to TRUE and OUT retains the value that it had in the execution cycle before the error occurred.

Further information

Product and service inquiries

Address any inquiries about the product to your local ABB representative, quoting the type designation and serial number of the unit in question. A listing of ABB sales, support and service contacts can be found by navigating to www.abb.com/searchchannels.

Product training

For information on ABB product training, navigate to new.abb.com/service/training.

Providing feedback on ABB manuals

Your comments on our manuals are welcome. Navigate to new.abb.com/drives/manuals-feedback-form.

Document library on the Internet

You can find manuals and other product documents in PDF format on the Internet at www.abb.com/drives/documents.

a1 (frozen)
PDF-A4
Created 2022-01-05, 07:21:01

www.abb.com/drives

3AUA0000127808G

© Copyright 2021 ABB. All rights reserved.
Specifications subject to change without notice.

3AUA0000127808 Rev G (EN) EFFECTIVE 2021-12-31