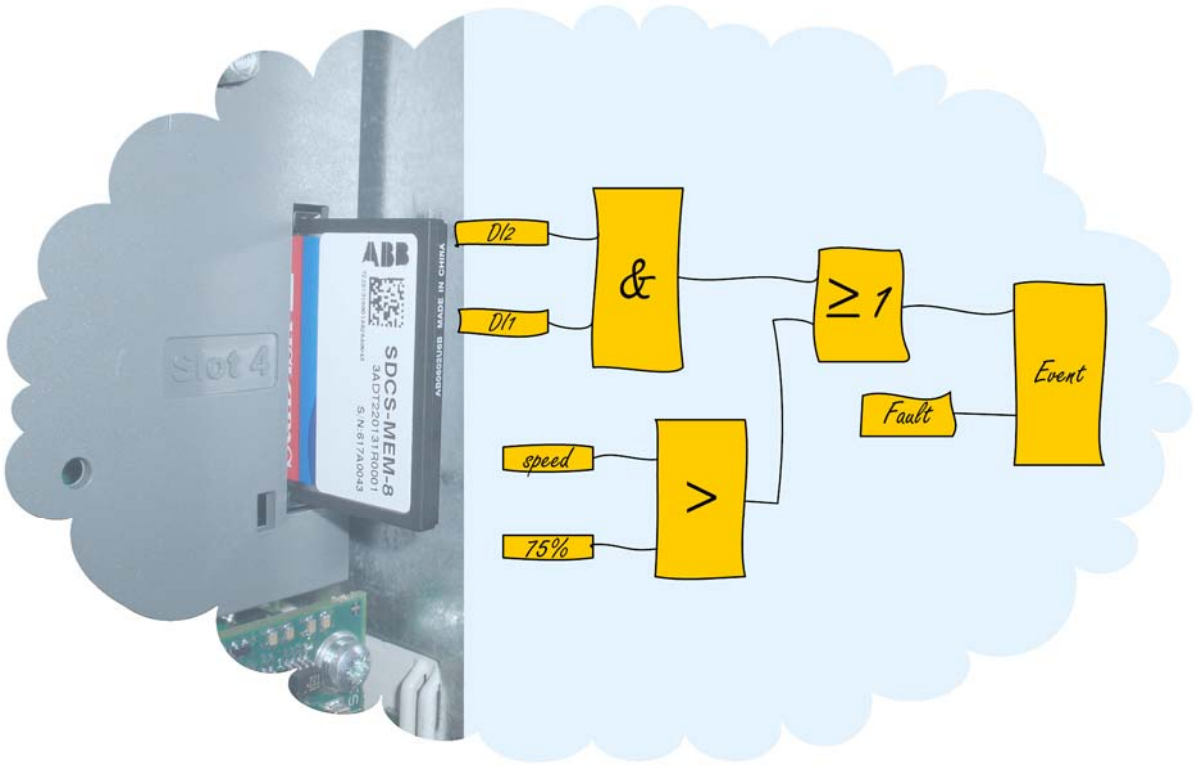
DCS800

Application Program

IEC61131-3 Target Installation Guide and Start-Up



DCS800 Drive Manuals

All the documents available for the drive system DCS800 are listed below:

		Language						
	Public. number	E	D	I	ES	F	CN	RU
DCS800 Quick Guide	3ADW000191	x	x	x	x	x		
DCS800 Tools & Documentation CD	3ADW000211	x						
DCS800 Converter module								
Flyer DCS800	3ADW000190	x	x	x	x	x	x	x
Technical Catalogue DCS800	3ADW000192	x	x	x	x	x	x	x
Hardware Manual DCS800	3ADW000194	x	x	p	x	x	x	x
Hardware Manual DCS800 update DCF503B/DCF504B	3ADW000194Z0301	x						
Firmware Manual DCS800	3ADW000193	x	x	p	x	p	x	x
Installation according to EMC	3ADW000032	x						
Technical Guide	3ADW000163	x						
Service Manual DCS800	3ADW000195	x	x					
12-Pulse Manual	3ADW000196	x						
CMA-2 Board	3ADW000136	p						
Flyer Hard - Parallel	3ADW000213	x						
Drive Tools								
DriveWindow 2.x - User's Manual	3BFE64560981	x						
DriveOPC 2.x - User's Manual	3BFE00073846	x						
Optical DDCS Communication Link	3AFE63988235	x						
DDCS Branching Units - User´s Manual	3BFE64285513	x						
DCS800 Applications								
PLC Programming with CoDeSys	CoDeSys_V23	x	x			x		
61131 DCS800 target +tool description - Application Program	3ADW000199	x						
DCS800 Crane Drive								
DCS800 Crane Drive Manual suppl.	3AST004143	x						
DCS800 Crane Drive Product note	PDC5 EN REVA	p						
DCS800 Winder ITC								
DCS800 Winder Product note	PDC2 EN	x						
DCS800 Winder description ITC	3ADW000308	x						
Winder Questionnaire	3ADW000253z	x						
DCS800-E Panel Solution								
Flyer DCS800-E Panel solution	3ADW000210	x						
Hardware Manual DCS800-E	3ADW000224	x						
DCS800-A Enclosed Converters								
Flyer DCS800-A	3ADW000213	x						
Technical Catalogue DCS800-A	3ADW000198	x						
Installation of DCS800-A	3ADW000091	p						
DCS800-R Rebuild System								
Flyer DCS800-R	3ADW000007	x	x					
DCS800-R Manual	3ADW000197	x						
DCS500/DCS600 Size A5...A7, C2b, C3 and C4 Upgrade Kits	3ADW000256	x						
Extension Modules								
RAIO-01 Analogue IO Extension	3AFE64484567	x						
RDIO-01 Digital IO Extension	3AFE64485733	x						
AIMA R-slot extension	3AFE64661442	x						
Serial Communication								
Drive specific serial communication								
NETA Remote diagnostic interface	3AFE64605062	x						
Fieldbus Adapter with DC Drives RPBA- (PROFIBUS)	3AFE64504215	x						
Fieldbus Adapter with DC Drives RCAN-02 (CANopen)								
Fieldbus Adapter with DC Drives RCNA-01 (ControlNet)	3AFE64506005	x						
Fieldbus Adapter with DC Drives RDNA- (DeviceNet)	3AFE64504223	x						
Fieldbus Adapter with DC Drives RMBA (MODBUS)	3AFE64498851	x						
Fieldbus Adapter with DC Drives RETA (Ethernet)	3AFE64539736	x						
x -> existing p -> planned								
Status 10.2008								

Table of contents

Introduction to the guide

Chapter overview

The chapter gives general information about the documentation.

Compatibility

The PC program CoDeSys is the hardware independent IEC 61131-3 programming system under Windows for creating controller applications. ABB DC Drives is using this programming system for DCS800.

DCS800 target files and DCS800 libraries are necessary for developing a software application. With functions out of DCS800 libraries the software application can be connected from and to firmware parameters and firmware signals.

Additional libraries contain further special function blocks, which simplify creating applications.

Safety instructions

WARNING! The safety instructions for the appropriate DCS800 DC Drive model must be studied carefully before doing program modifications or starting any work with the unit.
Neglecting the safety instructions may cause damage to the equipment, physical injury or death.

Especially when doing software applications, read also the software function specific warnings and notes before changing the default settings of the function. For each function, the warnings and notes are given in the Firmware Manual in the subsection describing the related user-adjustable parameters.
The developed application should never disable safety functions like E Stop, Off or like monitoring functions.

Reader

The reader of the manual is expected to:

- know the standard electrical wiring practices, electronic components and electrical schematic symbols.

About this manual

The documentation should be used together with DCS800 firmware manual. The firmware manual contains the basic information for the drive parameters. This documentation gives more detailed information about IEC programming for DCS800:

- How to install PC program CoDeSys
- How to install target files
- How to install the Memory Card
- How to start an application
- How to use special features of DCS800
- How to download an application into DCS800

Related publications

The user documentation of the drive also includes:

- Firmware manual (3ADW 000 193)
- Hardware manual (3ADW 000 194)
- CoDeSys manual (3S-software)

Glossary

The following terms and abbreviations are used in this manual:

Binary code	Compiled application program code for downloading into DCS800. The extension is *.PRG
CoDeSys	It is one of the most common IEC 61131-3 programming systems for PLCs and industrial controllers. All five programming languages of the standard are supported.
ControlBuilder DCS800	Software package of ABB on CD ROM; contains programming tool CoDeSys, plus required DCS800 target and plus DCS800 Drive Library.
DCS800 template	Basic application software for DCS800.
Download	Transfer parameters or program from user interface equipment (PC or panel) to DCS800.
DWL	Drive Window Light is a personal computer based commissioning and maintenance tool for ABB Drive products.
Project	Application program of CoDeSys
Upload	Transfer parameters or program from DCS800 to user interface equipment (PC).

Installation of CoDeSys Program

Chapter overview

The chapter describes the installation of the IEC61131-3 PC program CoDeSys including the DCS800 target and the DCS800 basic library.

Requirements

PC system requirements

- Pentium II, 500MHz
- 128 MB RAM (recommended 256 MB)
- 100 MB hard disc required
- Windows 98 / NT 4.0 / 2000 / XP / Vista
  (MS Internet Explorer 5.0 or later required)
- CD ROM drive
- Free serial port (RS232 connection to DCS800)

DCS800 requirements

- Memory Card SDCS-MEM-8
  (Compact Flash, specially formatted for ABB DCS800)

CoDeSys version and further updates

Use always the newest CoDeSys software which is released for ABB DCS800.

Before a newer software version is installed, please make sure that all used targets are released for this newer version.

Start installation from Tools CD ROM

DCS800 software tools are delivered with every drive in the “DCS800 Customer CD ROM.”

CoDeSys software program will be installed automatically together with other DCS800 software tools.

If the automatic installation is used, the following settings are done automatically for DCS800 drive. In this case skip the following installations descriptions, please.

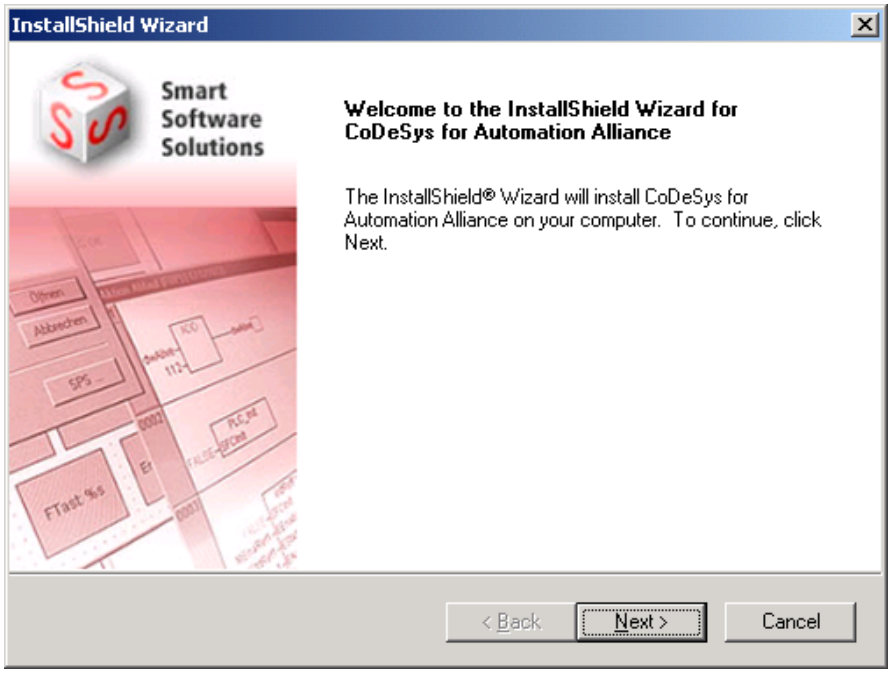
If a manual installation is used, please follow the instructions and hints.

Description

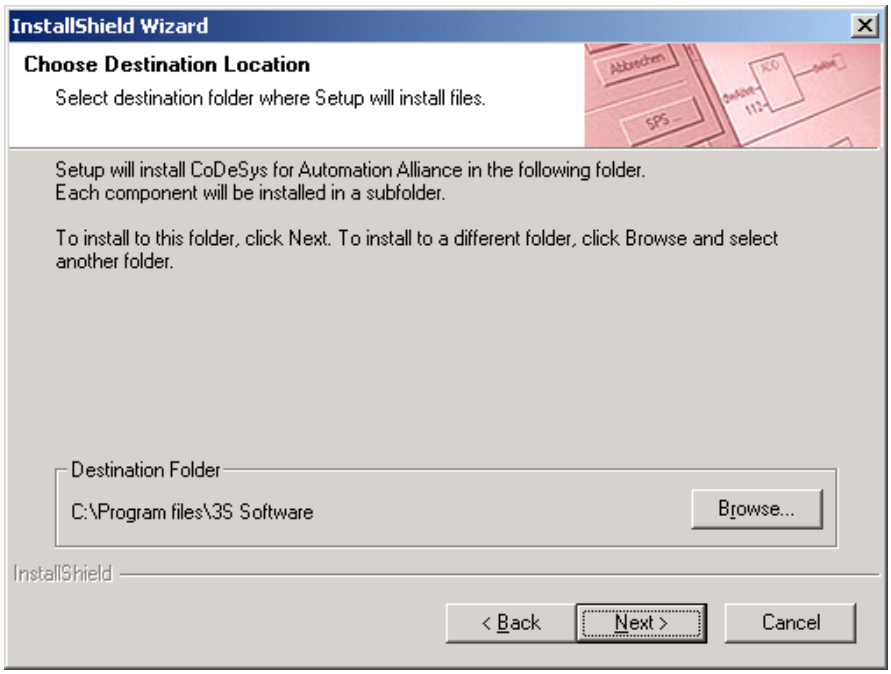
The description shows the most important windows of installation

Select a language for the setup:

Press **OK.**

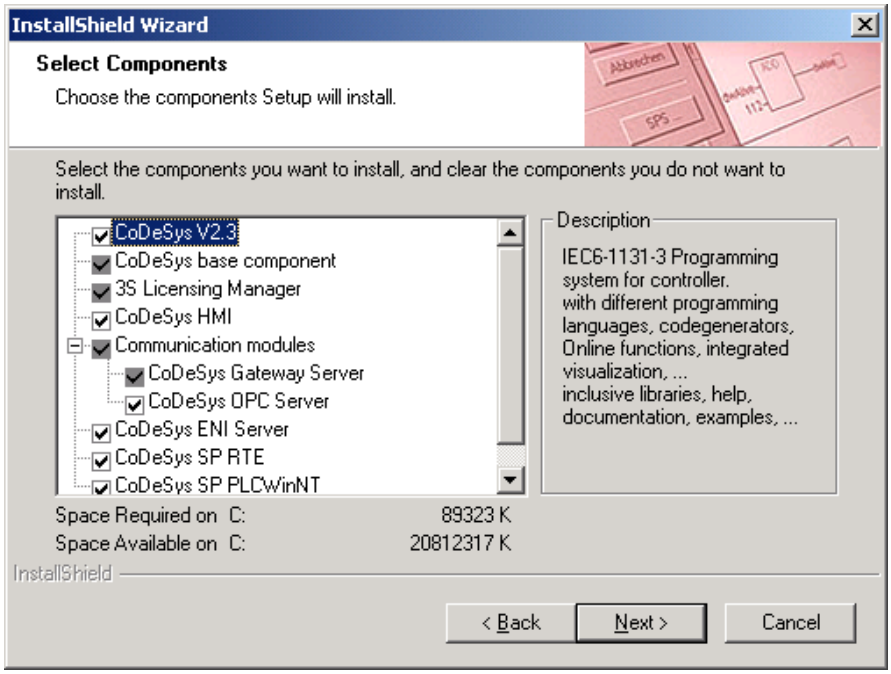


Press **Next**

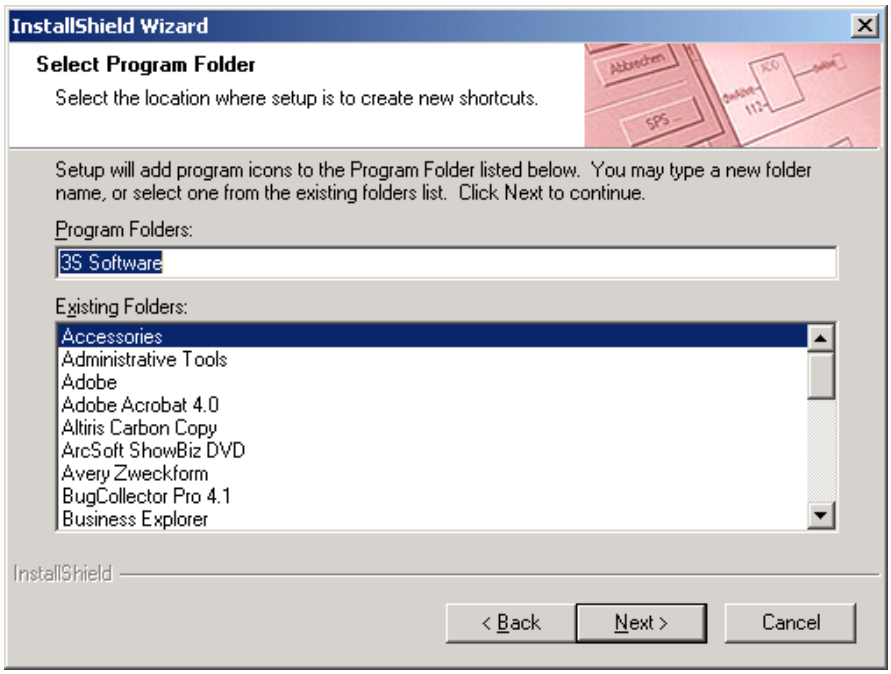


It is recommended to use the default “Destination Folder”. All further instructions are described using this default folder.

If another folder is desired, then change via “Browse…”, but remember always that you are using another folder.

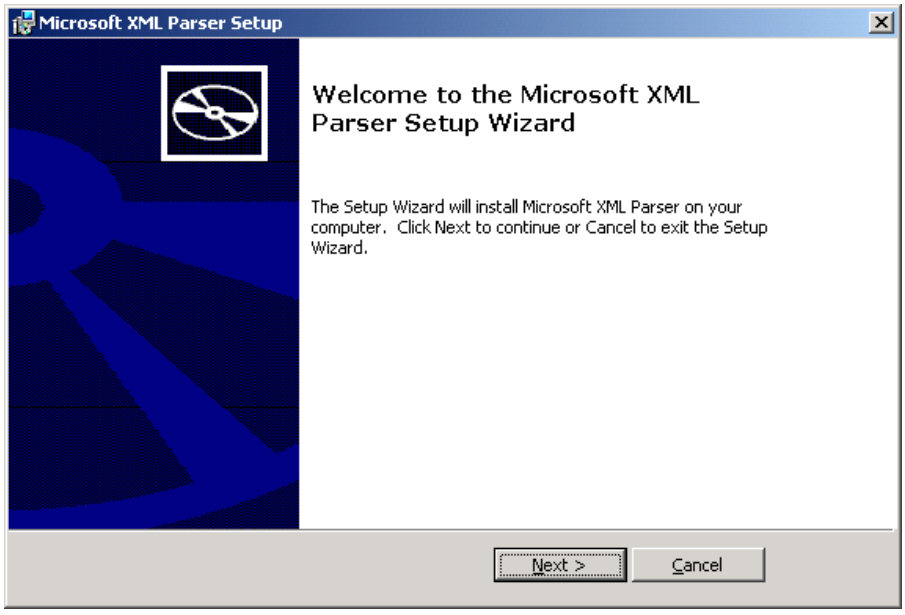


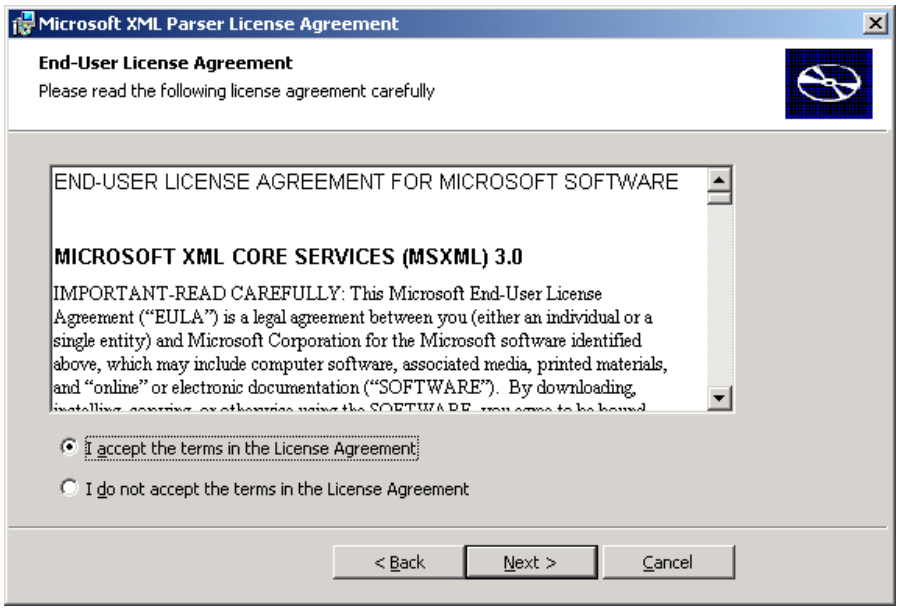
Please use the default settings.



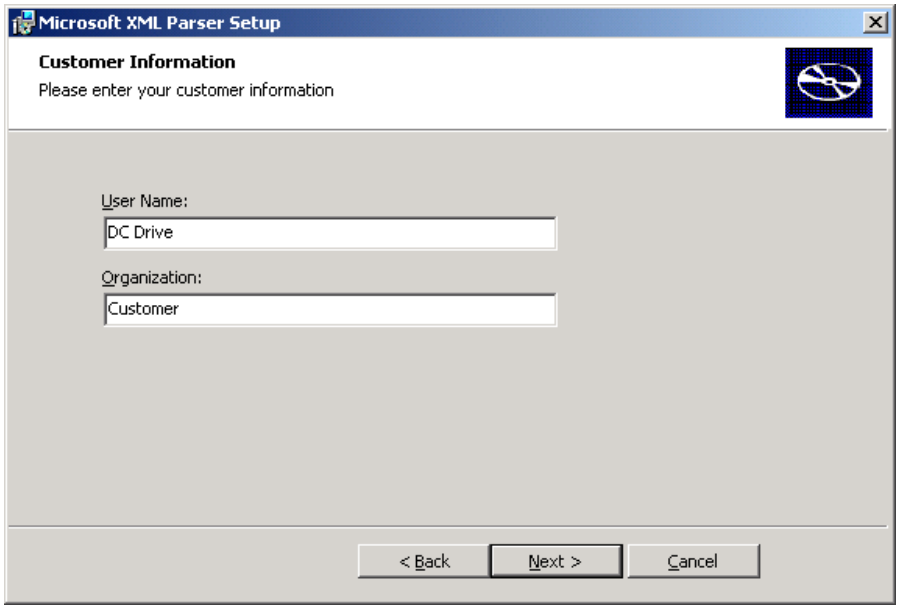
Accept or change the proposed program folder and then press **Next**.

During installation the setup will also install Microsoft XML Parser. Please follow these instructions.

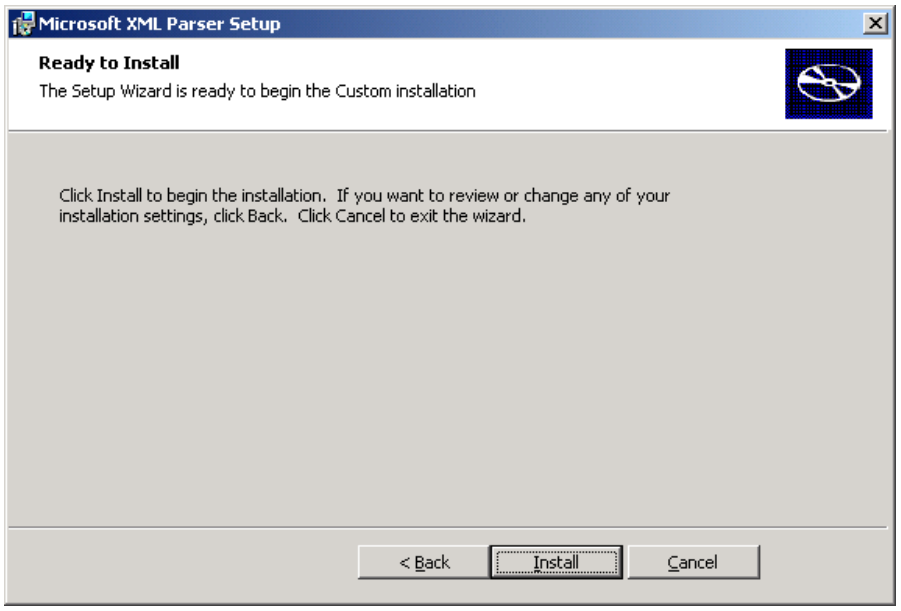




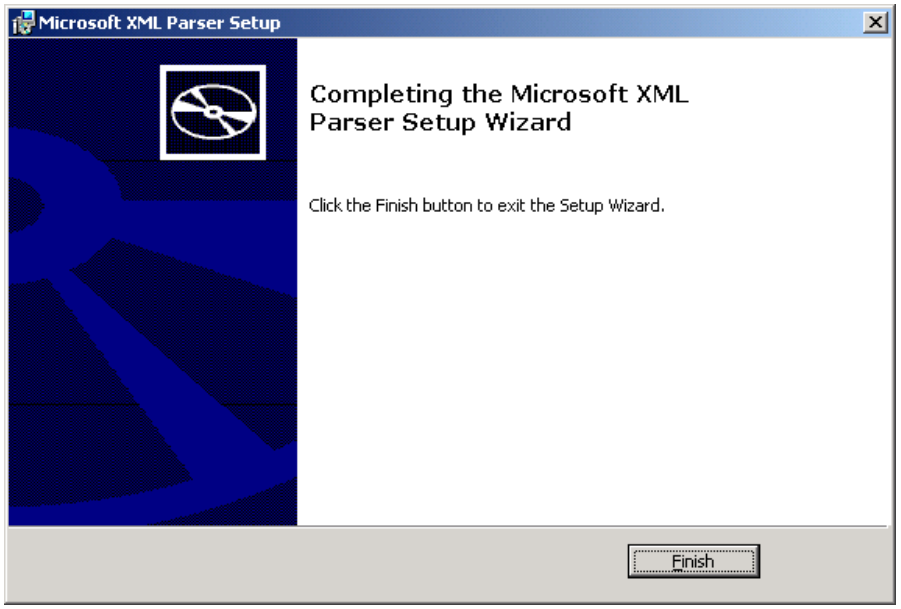
Read the end-user license and select "I accept …"
Press **Next** …



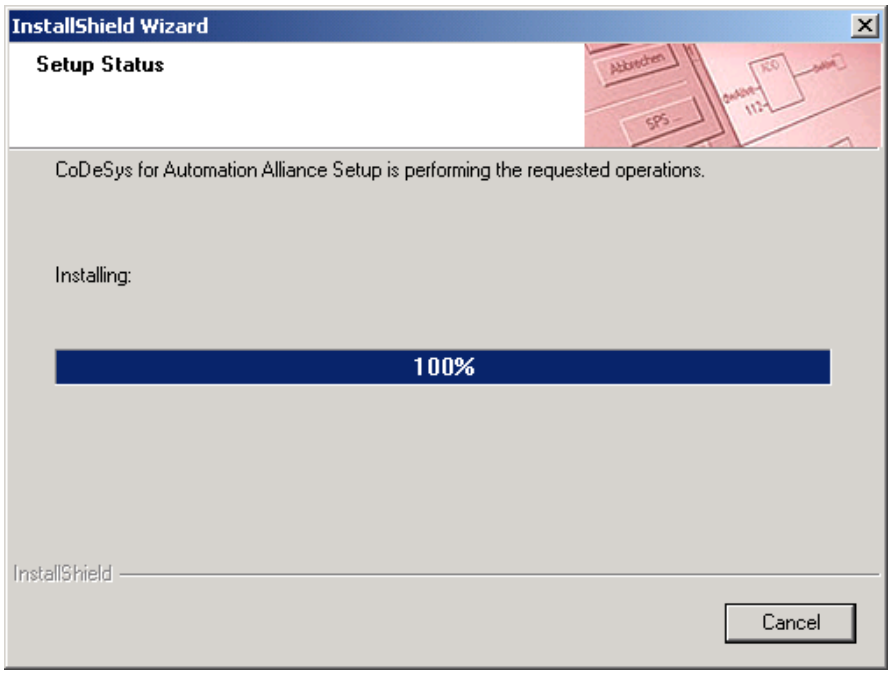
Write your name and your company and then press **Next**.



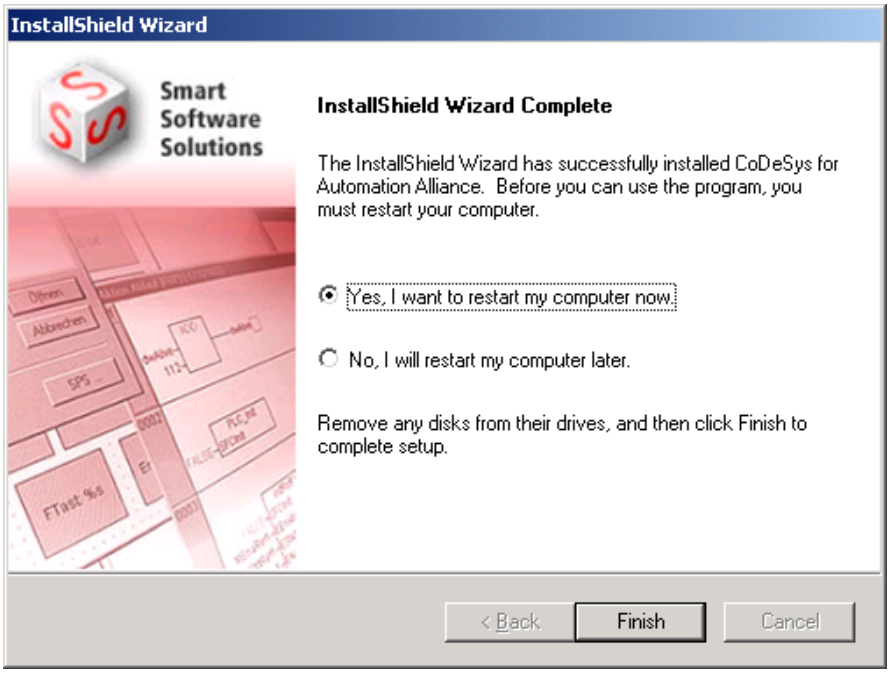
After pressing **Install** the program will be installed.



Press **Finish**.



There is no need to buy a license for DCS800 because it’s included in drive package.



Before starting the program a restart of PC is necessary.

Installation of DCS800 Target

Chapter overview

The CoDeSys programming software can be used with several PLC manufacturers. The target file defines the difference between the different PLC types.
The first step after opening CoDeSys software is to select the target for the PLC system. The target file includes information about the processor CPU, the download system and the released functions of the CoDeSys software tool. If the wrong target file is selected, it will not be possible to download anything to the PLC. (wrong compiler)

Please note:
When installing DCS800 Customer CD ROM, which includes CoDeSys and other PC tools, the DCS800 target is also installed. In this case, no further action is required.

When the CoDeSys PC software tool is installed with other targets (means other PLC types), the DCS800 target can be added manually. It is much easier to add one target than to replace the complete software PC tool.
Be sure to use a CoDeSys software version which is able to work with DCS800. Find information about the actual software versions in the DCS800 release notice.

DCS800 Target

For DCS800 Programming, the target file must be installed in addition to the programming tool.

The target file consists of two parts.

- Target information for programming tool with the information on CPU type, cycle times, RAM and FLASH size.
  With the target information, CoDeSys calculates free memory space during compiling. The load of the cycle time is calculated during online connection.

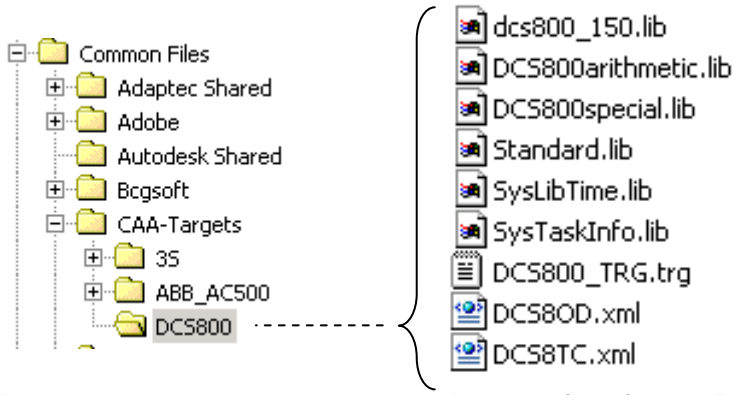
- The second part is the target (DCS800) specific library.
  These function blocks are specifically designed for DCS800 application programming.

Install the target manually

The automatic installation from the DCS800 Customer CD ROM installs the target files in the following directory:

Program Files \ Common Files \ CAA-Targets \ DCS800

For manual installation of target file, please manually copy the files into this folder.



Directory structure *Needed files for the DCS800*

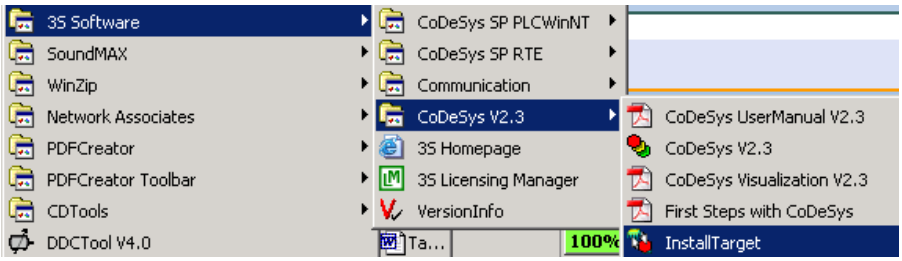
Note:

There is no need to save these files directly in this directory, but it is recommended to do this because other manufacturers use this directory also. This makes it easier to find the target files.

Settings inside CoDeSys

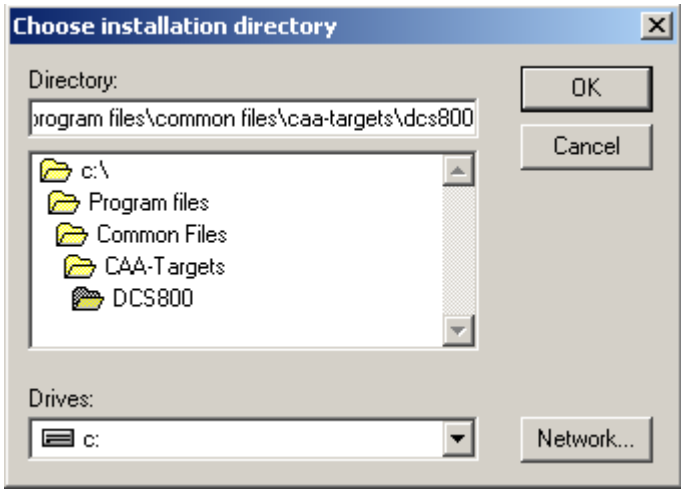
Install DCS800 target:

First, the target file for the DCS800 must be installed.
Select *Install Target* in the *3S Software* folder.



- Select the *Installation directory*:

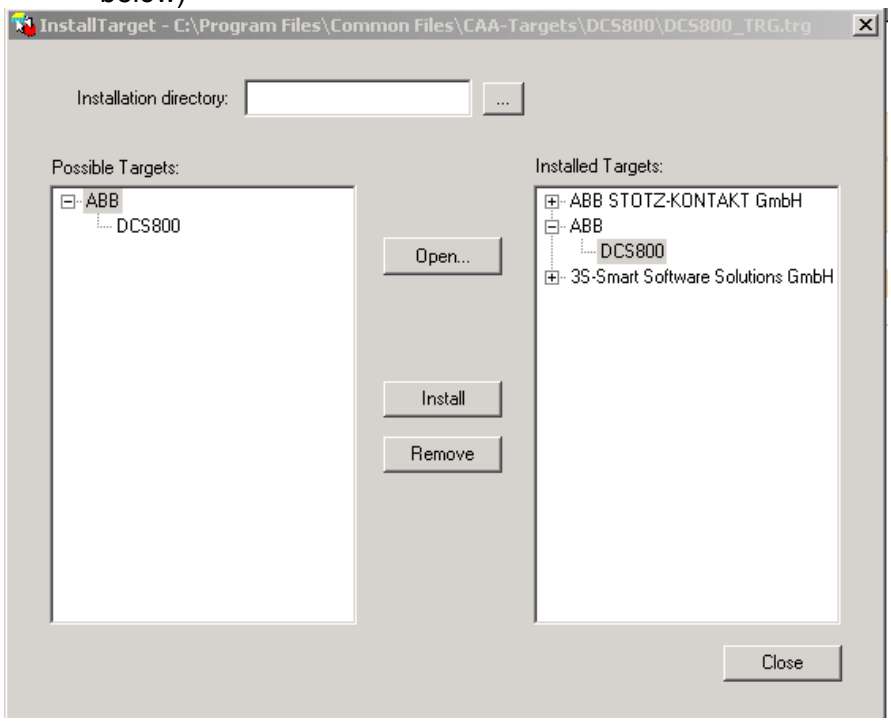
Program Files / Common Files / CAA-Targets / DCS800



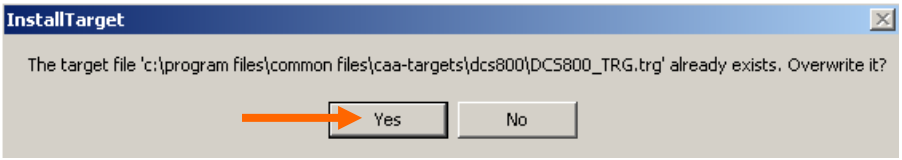
- Click to *Open* and select the file:

DCS800_Trg.trg

- Mark the text *ABB* and click the *Install* button (see picture below)



- If a message like this appears, respond with *Yes*, please.



Pointer to correct directories

For the DCS800, further settings have to be done for:

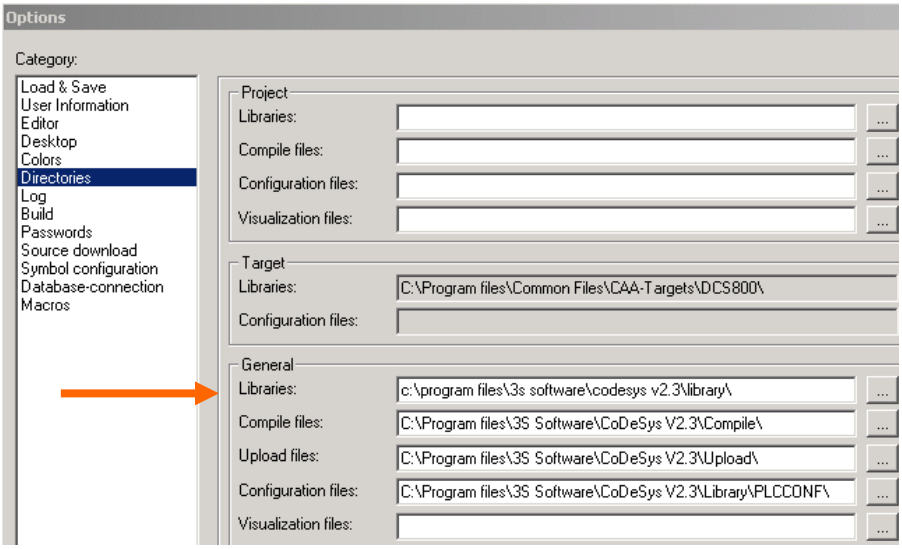
- Libraries for the DCS800
- Pictures for function blocks

Proceed as follows:

- Open CoDeSys software tool
- Click on menu *Project* submenu *Options*

Target Libraries:
Set the pointer of the target to the directory where the library files (*.LIB) are located.

General Libraries:
The bitmaps (*.BMP) for the function blocks have to be linked to the directory with the pictures.



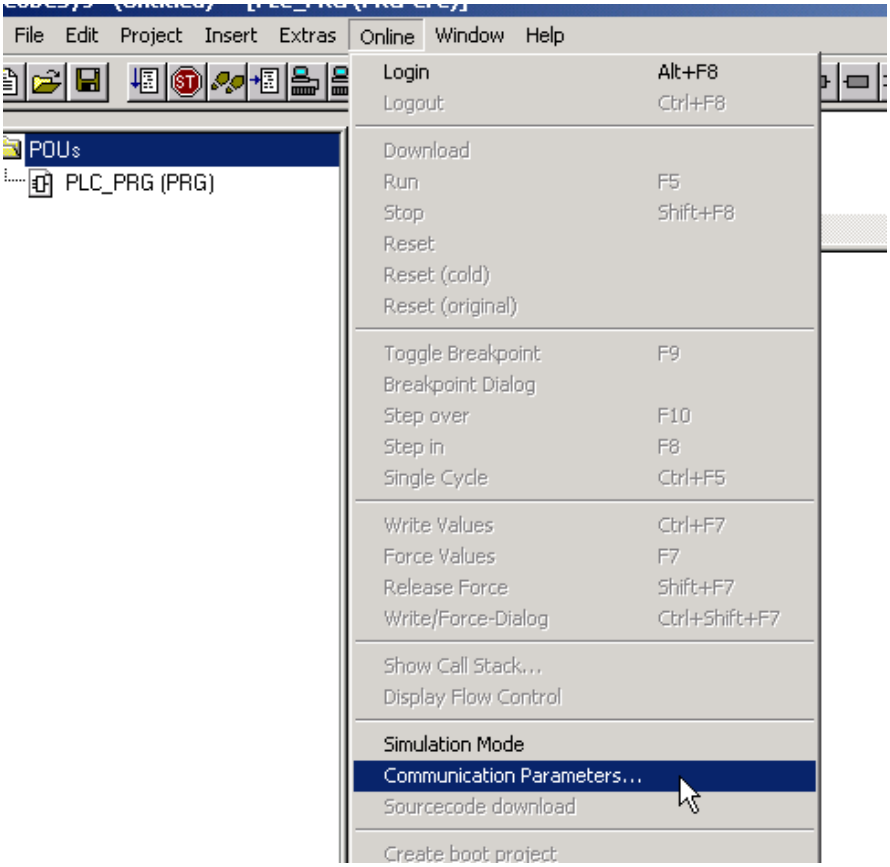
Setting Communication Parameters

Chapter overview

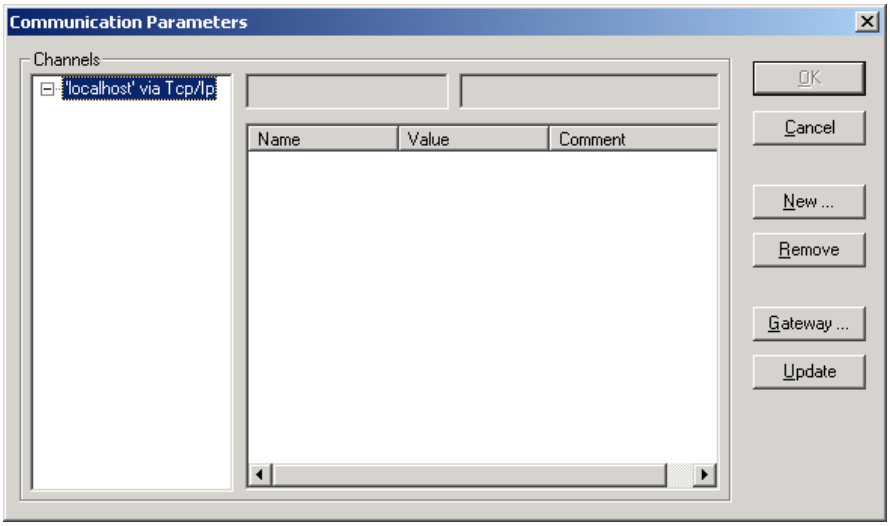
The chapter describes the settings of serial communication from PC to DCS800.

Create a new channel

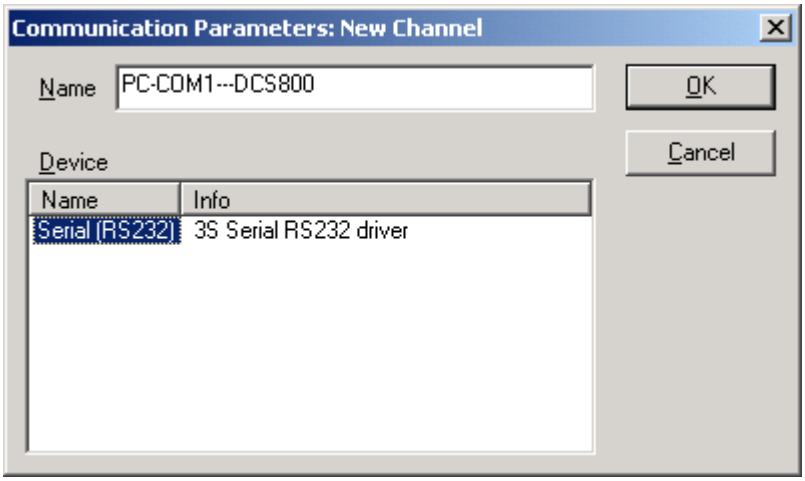
After starting CoDeSys software tool, please select "*Communication Parameters*" from the **Online** menu.



If no communication channel is available, a window like this will appear:



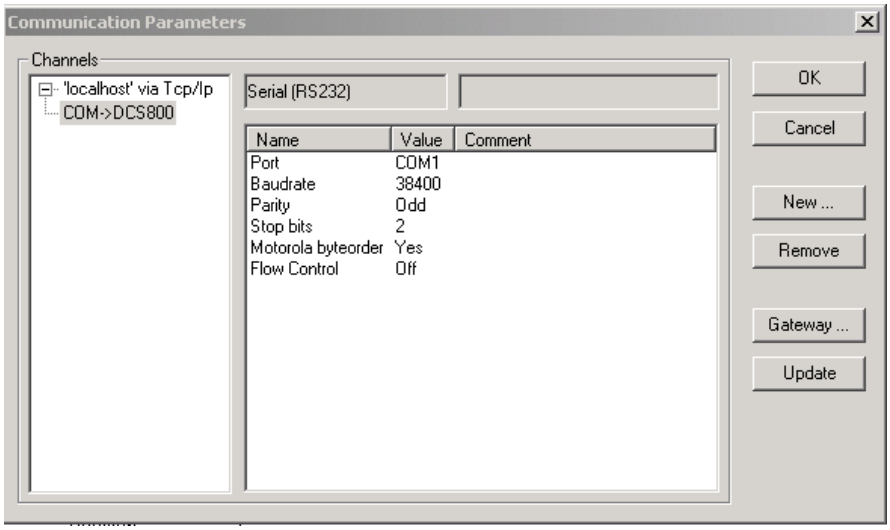
Press **New …** and define a name for the new channel;
e.g. *PC-COM1---DCS800*.



Select "*Serial (RS232)*" and press **OK**.

Attention:
If there is more than one device connection visible, then the DCS800 target is not installed correctly. In this case, check the target installation, please!

Communication Parameters



Change a setting by double clicking on the desired value. The values will change to the next available option each time it is double-clicked. The settings should be set as follows:

Port	COM1 (or another available port)
Baudrate	**38400**
Parity	**ODD**
Stop bits	**1**
Motorola byteorder	**Yes**
Flow Control	**OFF**

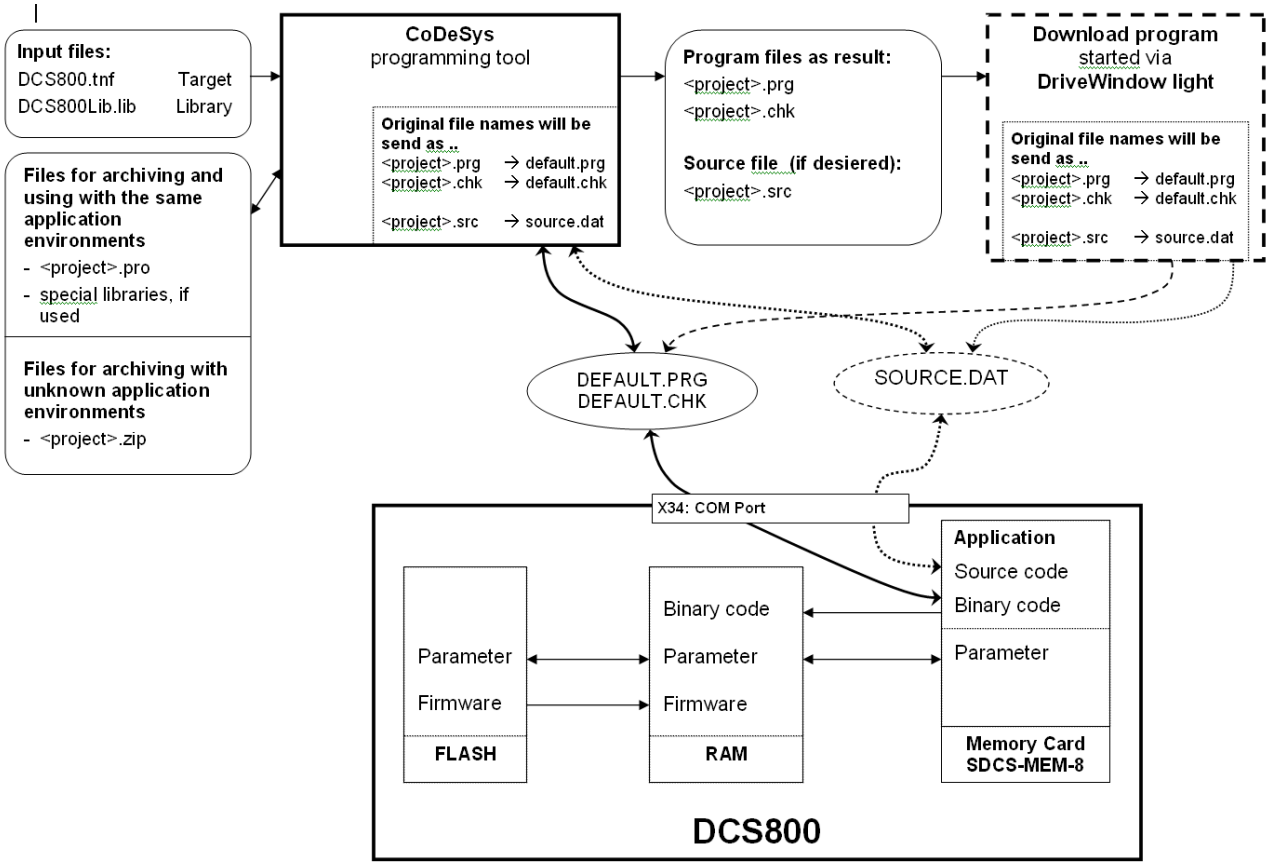
Press **OK** to save the settings.

Memory Card

Chapter overview

The chapter describes the handling with the ABB Memory Card.

Overview



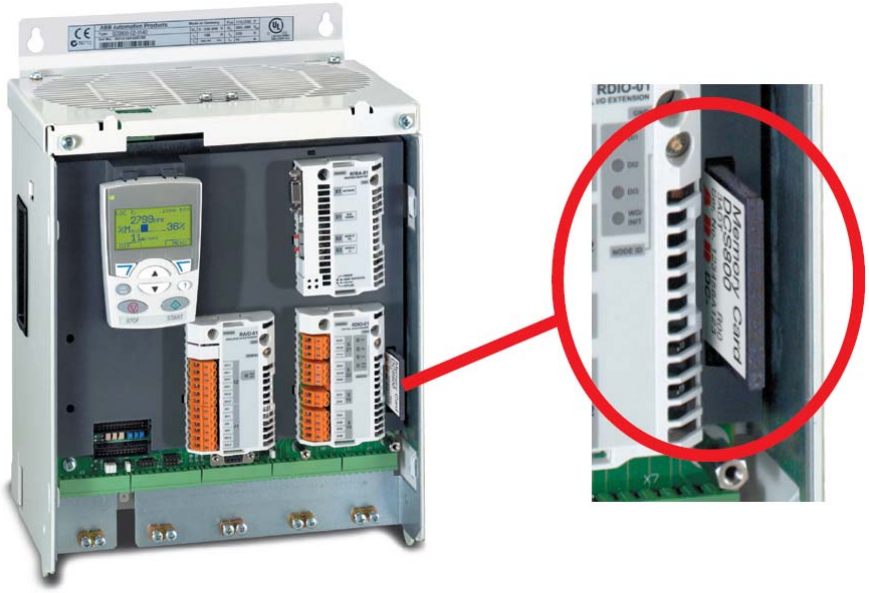
Requirements

The ABB Memory Card is used for application programming with DCS800. The ABB Memory Card is a **Compact Flash Card** (specially formatted!), which can be purchased from ABB only.
The application is stored on this ABB Memory Card. If desired, the source code and compiled code of the application can both be saved on it.

Please note:
This Memory Card is specially formatted and cannot be read or written to with PC equipment.
Writing via PC will permanently damage the ABB Memory Card making it useless to DCS800

Plug-In / Pull-Out

Switch-off the control power. (The control panel backlight should be off.)
Plug in or pull out the Memory Card and switch-on the power again.



Please note:
DO NOT plug-in or pull-out the ABB Memory Card with power on!

Downloading

Download an application via CoDeSys (Login of menu Online) or download the application via DriveWindow light (DWL).

Detailed information is available in chapter “Downloading into DCS800”.

Uploading

Binary code and source code of an application can be uploaded, unless it has been protected (locked).

Detailed information is available in chapter “Uploading from DCS800”.

Saving Binary Code

Application code will be written into the Memory Card by “Create boot project” with menu **Online**.

Detailed information is available in chapter “Downloading into DCS800”.

Once an application program has been enabled, the DCS800 will show alarm A142 (MemCardMiss) if the Memory Card is removed or is not working properly.
To clear the alarm and run the DCS800 without an application, the application program has to be disabled by setting parameter **P16.06** = **DisableAppl**.

Storing parameters

DCS800 parameter settings can also be saved on the Memory Card. Command at parameter ***P16.06 SaveToMemC*** saves the actual parameters on the Memory Card. ***P16.06 LoadFromMemC*** loads the saved parameters to the drive system.

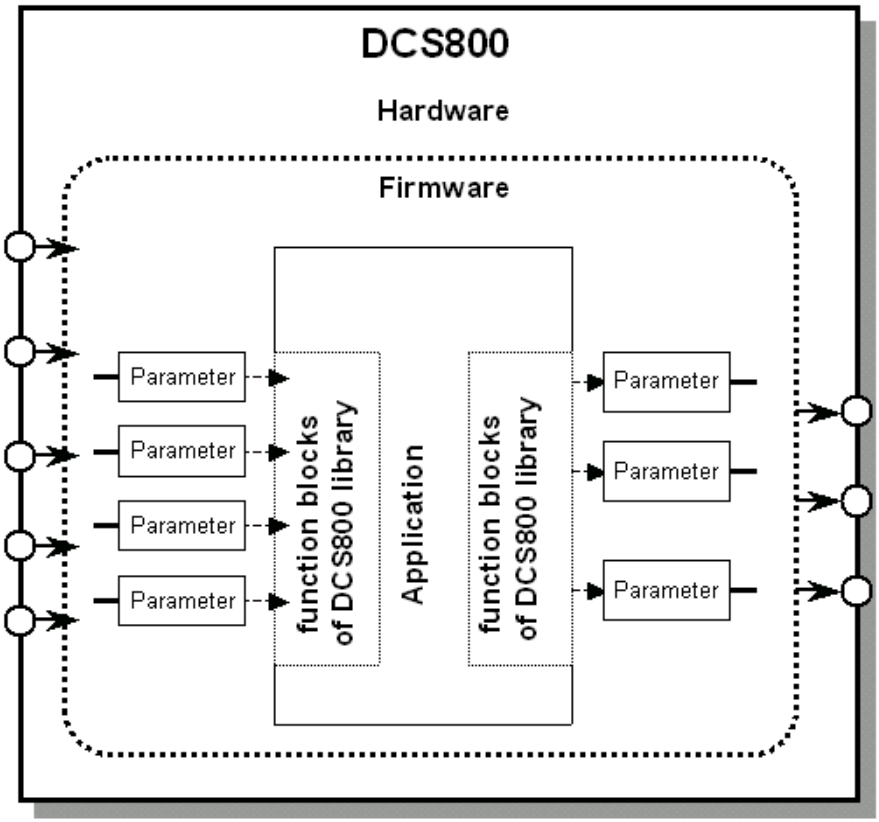
Start-Up Application

Chapter overview

This chapter describes how to create an application programming and describes important functions for the DCS800.

Interface between DCS800 and IEC

Once complete, the application software will be embedded in the firmware of DCS800. Signals for the application can be read from the firmware parameters. It is also possible to write to firmware parameters.
Basically the only possibility to implement an application program is to read parameters, do calculations or operations and write to parameters.



New project

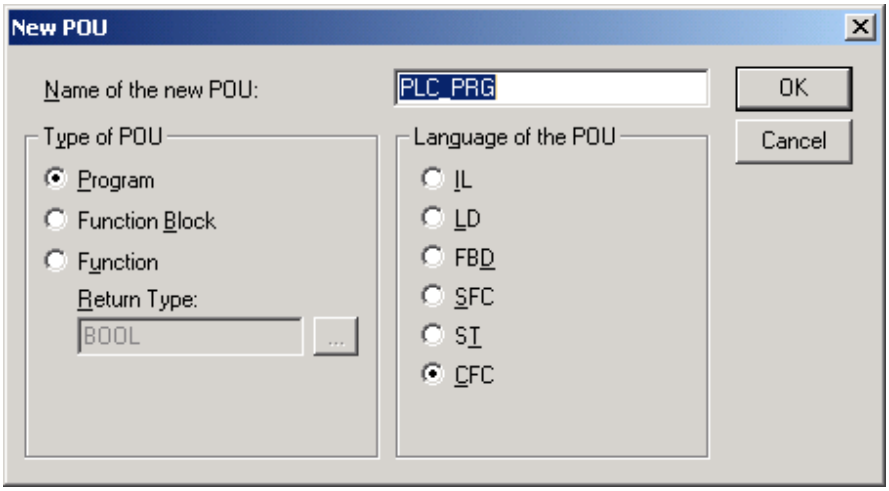
After starting the CoDeSys program, a new project can be started by selecting **New** from the **File** menu.

It can also be started with **New from template …** from the **File** menu, if a project with presettings has been saved before.

This chapter describes how to start a new project without presettings.

Select DCS800 as target. This will automatically select the correct compiler for DCS800.

The target setting contains the default setting. Without special knowledge please use these default values and press **OK**.



The default program is always called *PLC_PRG*.
Select the type **Program** and the desired language of your application.

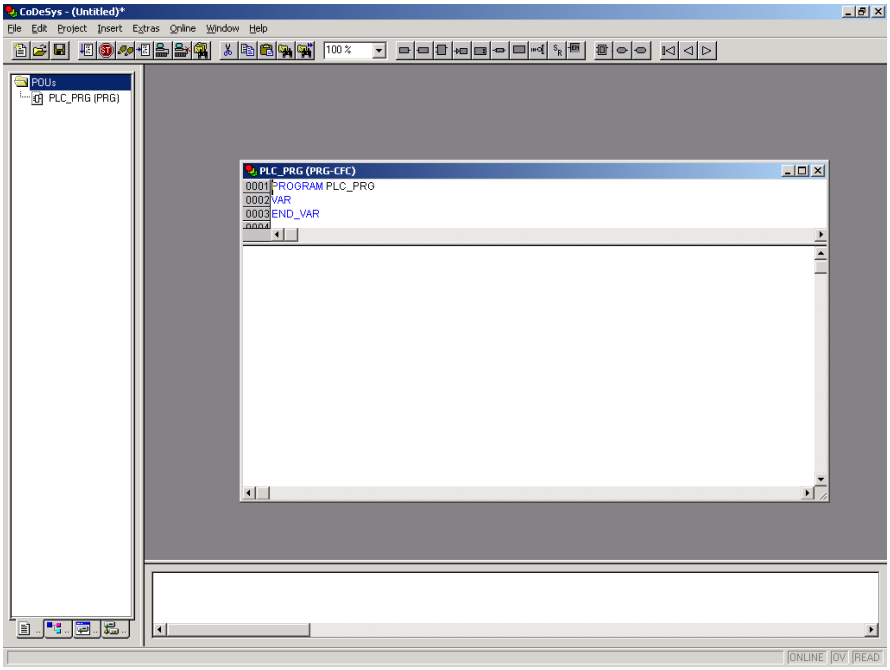
Available programming languages:

IL:	Instruction List
LD:	Ladder Diagram
FBD:	Function Block Diagram
SFC:	Sequential Function Chart
ST:	Structured Text
CFC:	Continuous Function Chart

Then press **OK**.

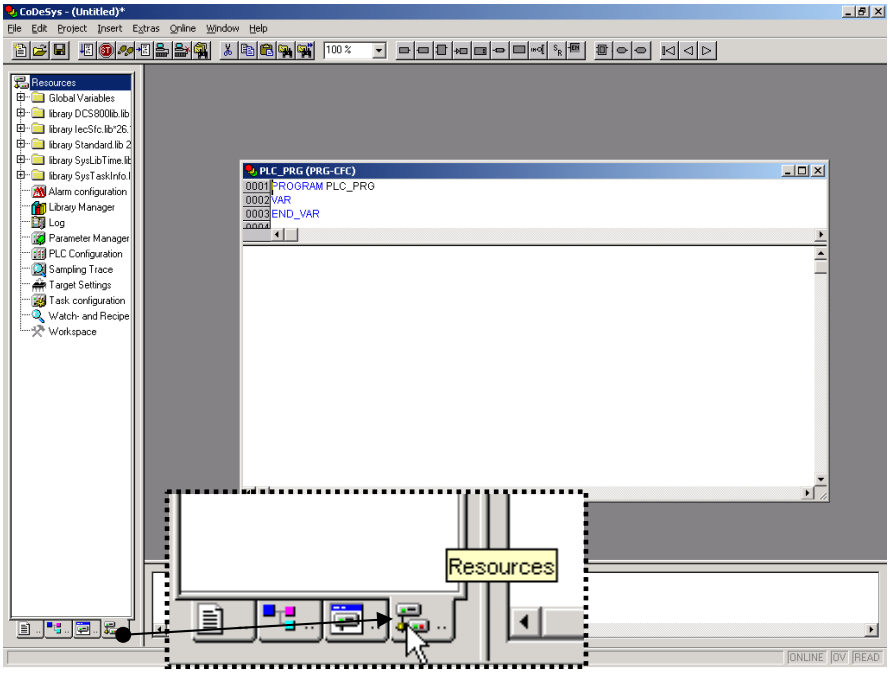
Attention:
Normally the main program *PLC_PRG* is required in each program, but DCS800 is a task controlled system and it, therefore, is possible to define another name for the main POU (**P**rogram **O**rganisation **U**nit).

A new program window will be opened which can be used for the new application program.



Library Manager

The *Library Manager* handles the administration of function blocks which are present in libraries. It is possible to add or remove libraries if needed.
Find the *Library Manager* in tab Resources:



Select the tab *Resources* and then *Library Manager*.

Install further libraries

CoDeSys software tool distinguishes between 3 types of boxes which can be used to create an application program:

- Operators (AND, OR, ADD, SUB, MUL, DIV,…)
- Functions (User specific)
- Function blocks (TON, TOF, …)

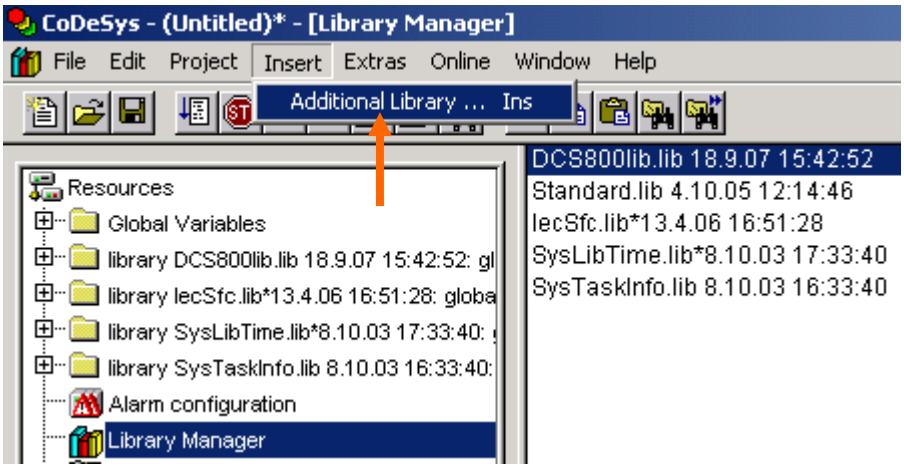
Note:
Operators are available independently from the selected target, but there are some operators which may not work with every target system!

Normally only the following libraries are added to a project in default status:

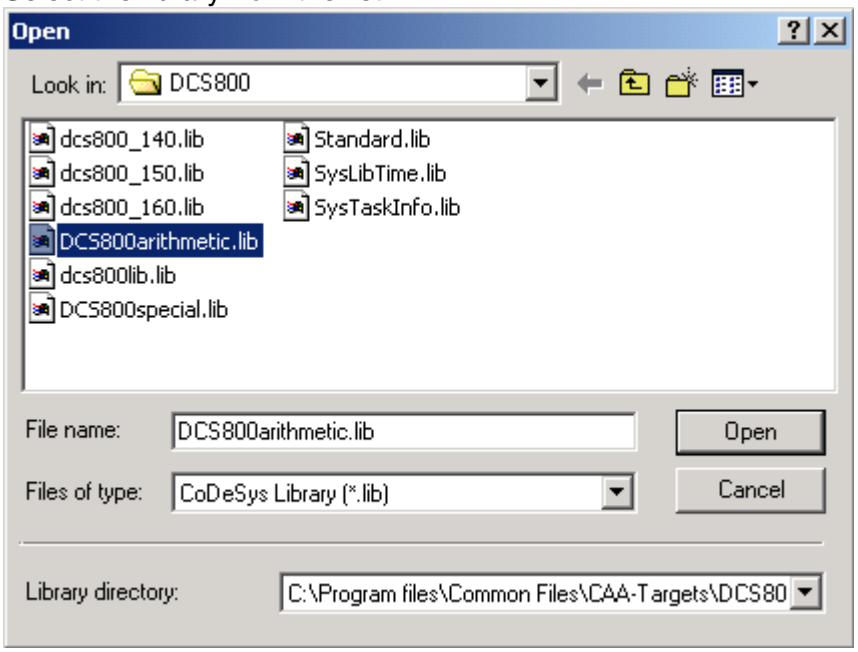
- DCS800lib.lib
- Standard.lib
- IecSfc.lib (not relevant for DCS800)
- SysLibTime.lib (not relevant for DCS800)
- SysTaskInfo.lib (not relevant for DCS800)

If more libraries are needed they have to be added manually:

- Change to the *Library Manager* in CoDeSys software tool (see picture)
- Select "*Additional Library*" from the **insert** menu



Select the library from the list



After the implementation of the libraries, they can be used like other libraries.

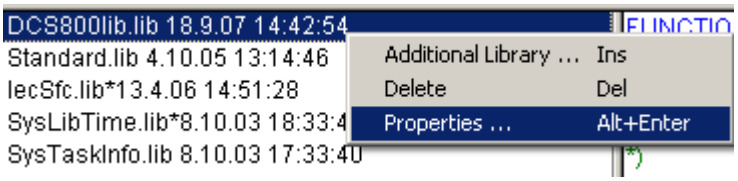
Please note:

It isn't allowed to implement libraries twice. The same rule applies for function blocks with the same name but perhaps in different libraries. This would cause error messages during compilation.

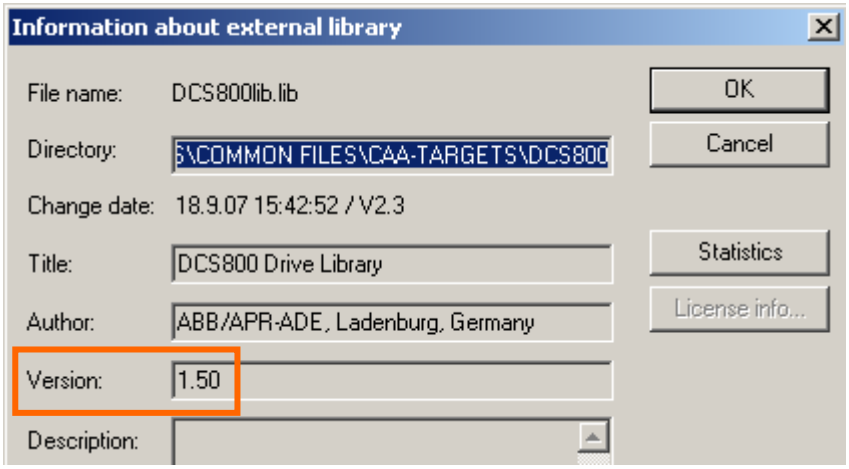
Library version

Software based libraries include a software version which has to be checked. Particularly the *DCS800lib* must be checked because the code for the function blocks is implemented in DCS800 firmware and both systems (DCS800 firmware and CoDeSys) must be compatible.

To check the library version please go to *Library Manager* and click with the right mouse button to a library.



Then select properties and the following window appears:



Attention:
It's necessary to check the library version of the DCS800 firmware (Parameter 4.13) and the version of the DCS800 library (right mouse click to the library, select properties).
The library will only work if the following rule applies:

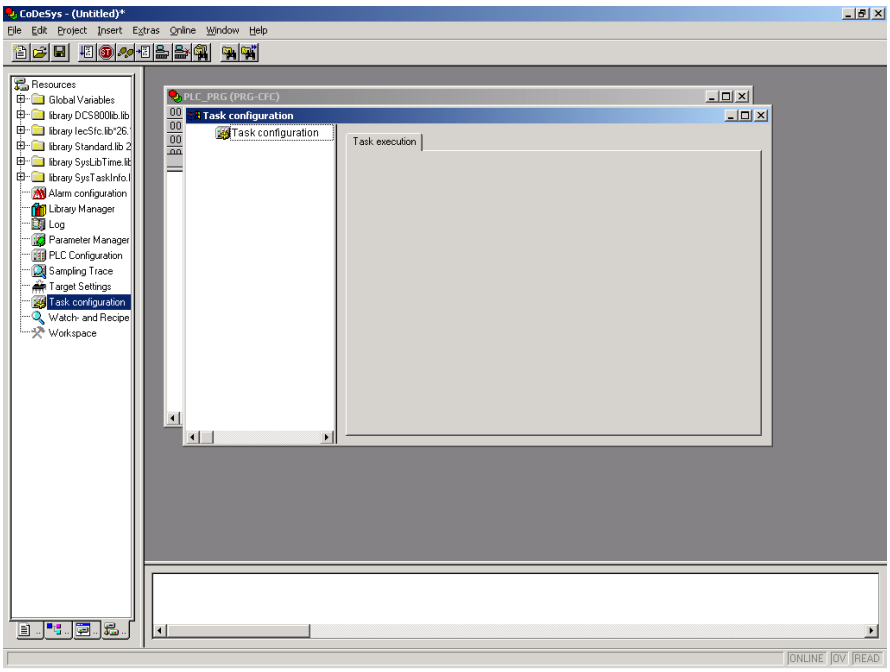
[DCS800 library version] ≤ [Para 4.13 of the DCS800 firmware]

Note:
The file name *DCS800lib.lib* is reserved for the DCS800 library.
If the DCS800 target is selected, the file *DCS800lib.lib* will be inserted automatically from DCS800 directory to *Library Manager*.

It is recommended to save the library files with the version name (e.g. *dcs800_150.lib*), copy this file and rename it to *DCS800lib.lib*.

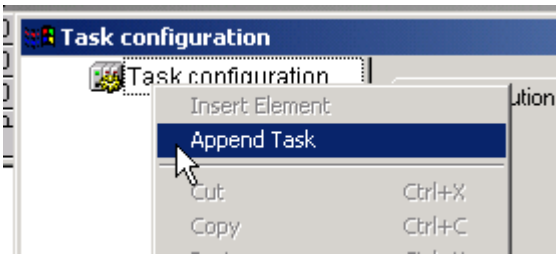
Task Configuration

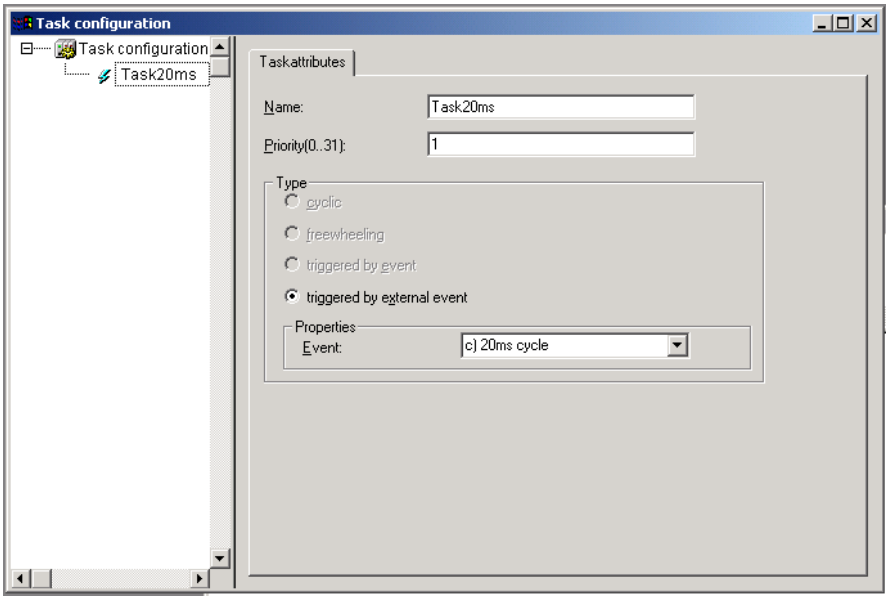
In tab **Resources** there can also be found the **Task Configuration**.



With double click the **Task Configuration** window will be opened.

Add a new task by a click to menu *Insert* and *Append Task*.





Define a name for the task attribute

The priority isn't supported from DCS800

Select event property

Available events:
Init, 5ms, 10ms, 20ms, 50ms, 100ms, 200ms, 500ms, 1000ms and some triggered by actions.

Application Source

Now create your application by using boxes from libraries and IEC operators.

Please note:

DCS800 firmware (parameter and signals) can handle only 16 bit types (BOOL, BYTE, INT, WORD). Real types must be converted before writing to a parameter or signal.

New Parameter Groups

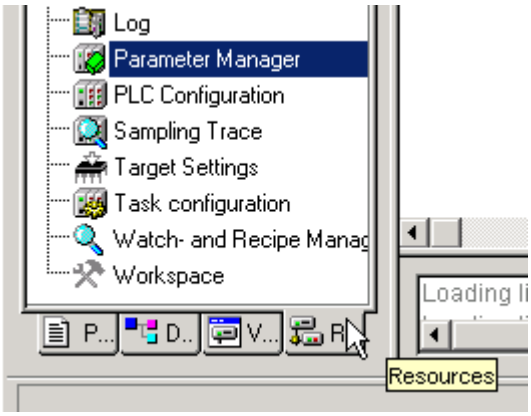
With application program, new parameters or signals can be created. The new parameters are handled from the control panel or PC tool like original firmware parameters.
The **groups 60 … 69** are reserved for new parameters created by the application program. It is strongly forbidden to create new parameters outside groups 60…69. This could cause problems with parameters or firmware.

Parameter Manager

To create new parameters, use the function “parameter manager.” The new parameters provide the following functions:

default
maximum
minimum
scaling / enumeration
volatile / non volatile

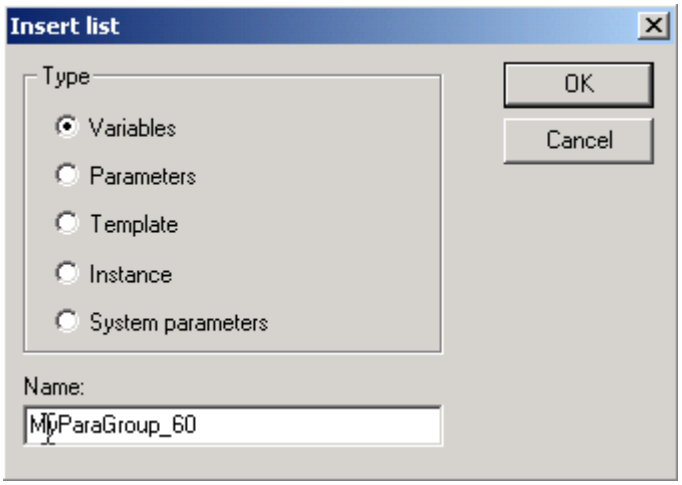
Start Parameter Manager in tab Resources



Create a new list

After starting, select **List…** in the **Insert** menu …

… the “Insert List” will be opened.



Select **Variables** (this is the only valid selection) and write a name for this list.

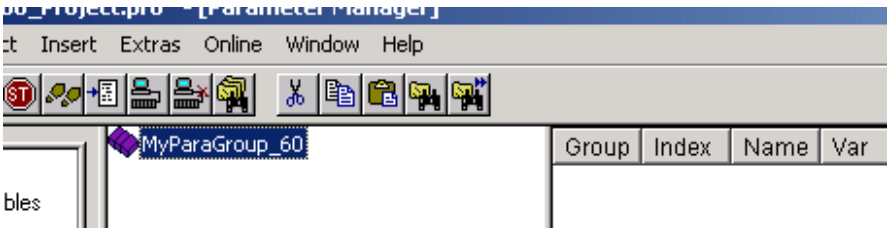
Please note:

The name of the list will also be shown as the name of the group in the parameter list, so it is recommended to insert a separate list for each parameter group!

It is <u>not</u> possible to select any other types, such as parameters, template and so on!

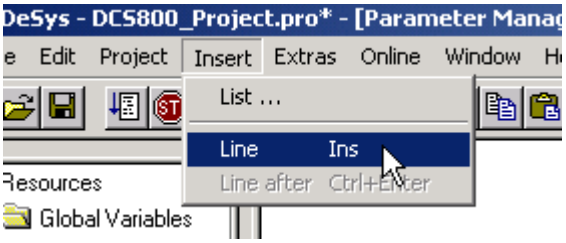
Then click **OK**

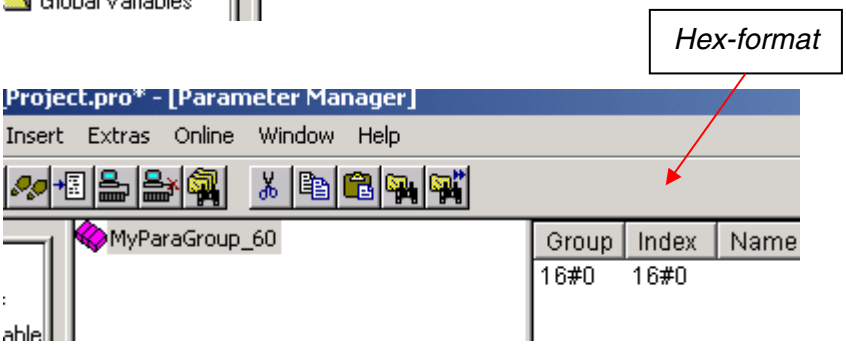
A new list is created.



Create Parameter

Mark a desired list (group) and insert a line.





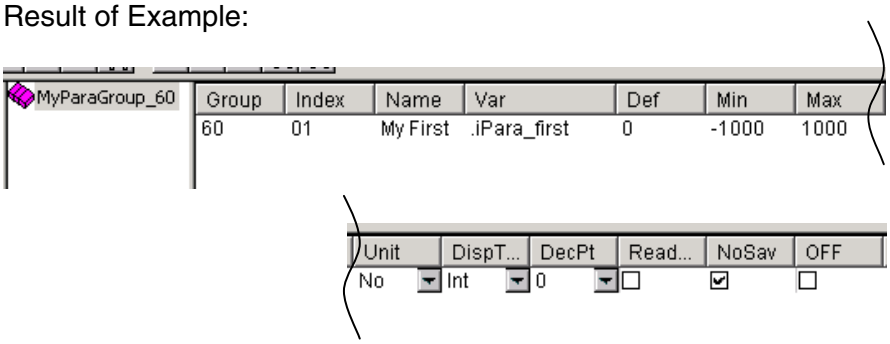
Click on a field of this new line, beginning with group, and insert the following data:

		Example:	
Group	group number of parameter.	60	as parameter **60**.01
Index	index number of parameter	01	as parameter 60.**01**
Name	Name of parameter, which will be used as the parameter name in the DCS800's panel and in PC tools. The name can be a maximum of 17 characters.		My First
Var	Name of Variable in the program. A dot before this variable means that it is handled as a global variable.		**.**Para_First
Def	Default value, which is set after downloading application.		0
Min	Minimal value		-1000
Max	Maximal value		1000

Note:
Global variables can be used in several programs. For new created parameters it is necessary to define global variables. For parameters there are only the types INT, BOOL and BYTE possible. Fast actualisation of parameters is only possible with type integer (INT)!

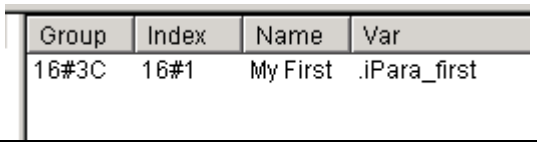
Unit	Unit of this parameter, which will be shown in DCS800's panel and the PC tool. *Possible selections:* *No, A, V, Hz, %, s, h, rpm, kh, deg C, lb-ft, mA, mV, kW, W, kWh, deg F, Hp, MWh, m/s, m³/h, dm³/h, Bar, kPa, GPM, PSI, kHz, days, min, ms, %/ms, mH, mOhm, Ohm, kAh, deg, rpm/s, ppr, Nm, Nm/s, N, kN, N/s, mm, m, m/min, m/mins, m/s², kgm², kg/dm³, lb-ft², ft/s², ft/min, ft, in, ft³/h, ft³/min, ft³/s, gal/h, BTU/h, 1/min, rev*	No
DispTyp	Type of parameter, in which the value has to be shown. *Possible selections:* *Char, Byte, Int, Word Enum, Hex, Bin.*	Int
DecPt	Digits after decimal point. *Possible selections:* *0, 1, 2, 3*	0
ReadOnly	Parameter can only be read. Parameter selected as read only can only be used as signals.	not selected
NoSav	Parameter will not be saved on the flash.	not selected
OFF	Before changing the value the drive has to be switched off.	not selected

Result of Example:



Please note:

After e.g. parameter manager is reopened, group and index will be shown in Hex format as indicated by “16#.”
Group: 60 = 3C; Index: 01 = 1.



Identification

CoDeSys offers the possibility to identify the developed application by means of “Project Info”.
The first 12 characters of the fields “Title” and “Version” are shown in the drives parameter group 4 “Information.“ “Title” is written to *ApplicName* (P4.03) and “Version” is written to *ApplicVers* (P4.12). This allows the identification to be read by the control panel and the PC tool.

Often, at the time of application development, the need of a long-living identification is not thoroughly considered.
Sooner or later, however, the author of the application becomes unknown. This is not a problem as long as the drive is running, but in case service is required, a proper name and description of the application will help to avoid serious problems and delays.

Registration

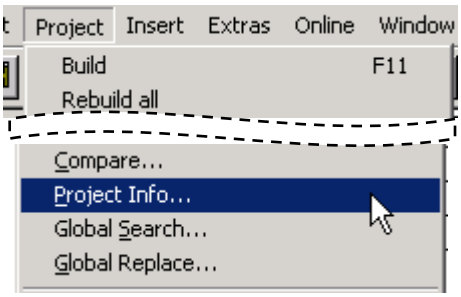
Because of this, we have developed these identification standards:

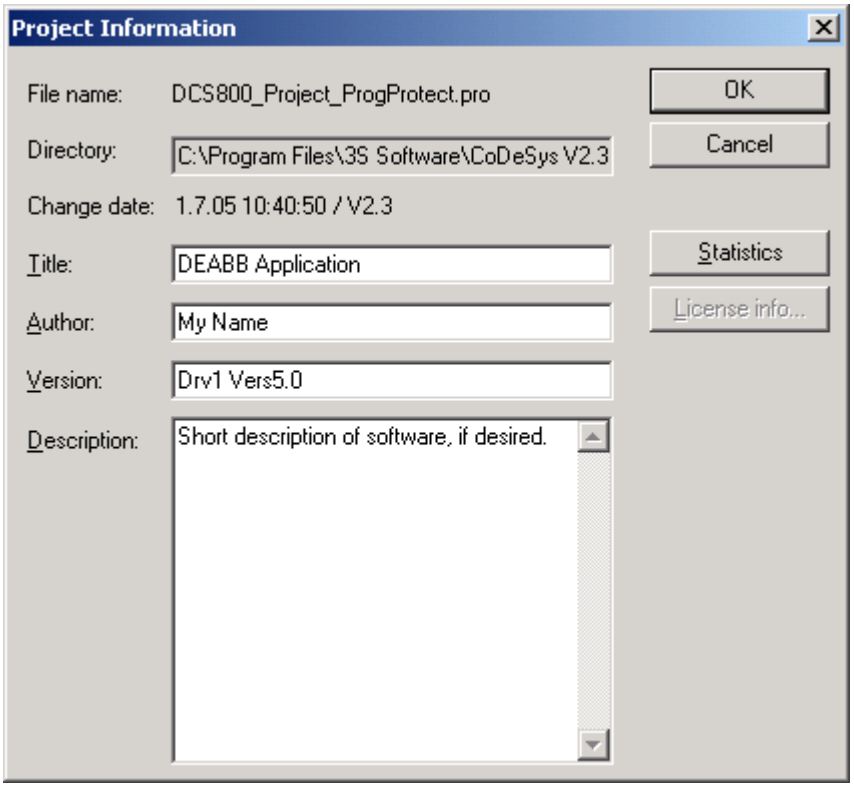
- The first 2 characters of the field title contains the international country name code of your country; e.g. US for United States or CN for China or DE for Germany.
- the next 4 characters of the field title are reserved for the shortened name of your company; e.g. USABB_ or DECUST
- all following characters can be used at will.

Please register the first 6 characters of your personalised identification with ABB DC drives factory, so we will be able to collect information for each registered application.
Additionally, we can inform you about any news about DCS800 application programming.

Setting

The project information window can be found under “**Project Info ...**” in the **Project** pull down menu:





The first 12 characters of the fields “Title” and “Version” will be shown in parameter group 4:

Title:	**ApplicName**	**(P4.03)**
Version:	**ApplicVer**	**(P4.12)**

Protection

Projects are normally saved on the PC. Projects may be protected with two levels of passwords for read or write protection.

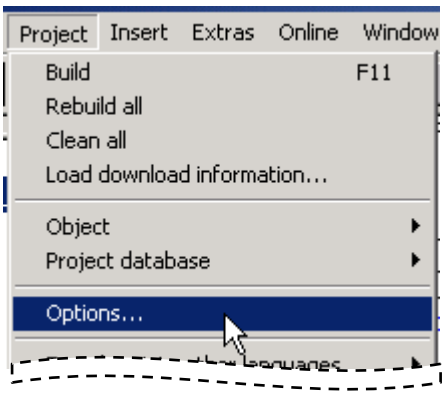
The binary code must be downloaded into the DCS800's Memory Card (ABB MEM-8). The source code can also be downloaded, if desired.

Password protection protects against illegal copying.

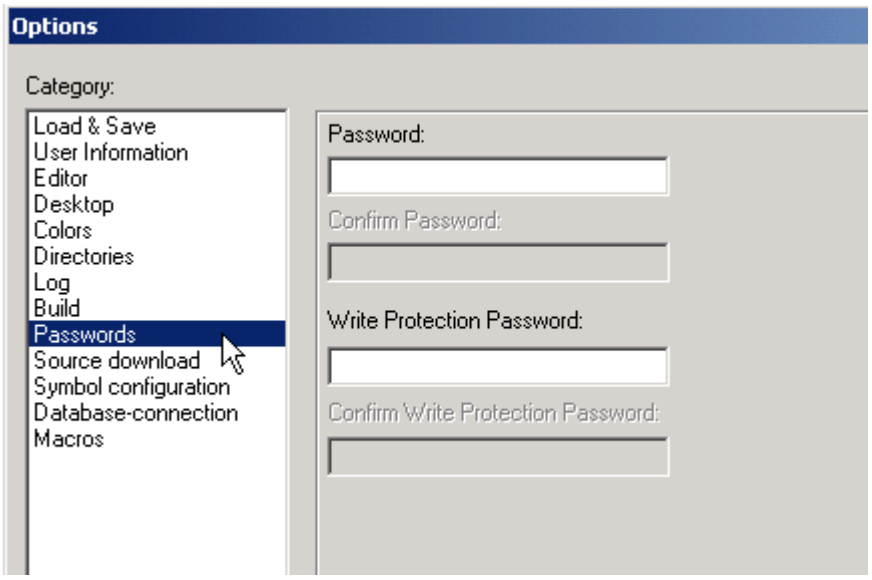
However, please be aware of these consequences:
- With an unknown or forgotten password, you will never be able to view this application again.
- With a protected project code, you can never duplicate an existing Memory Card which could be important if service is needed.

Password for project files

The CoDeSys program supports two levels of password to protect the program source.



Select **Options ...** from Project menu.



Select **Passwords** of menu Category.

Type in a password!

Password to protect the source files.
The source files can not be opened without this password.

Write Protection Password
for write protection of the source files.
The source files can be opened without this password, but cannot be modified.

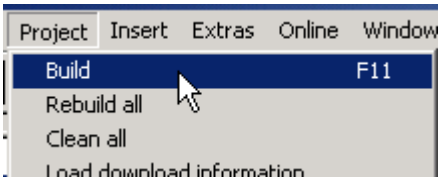
Please note:
It is the responsibility of the programmer to maintain records of all passwords. Failure to do so could result in loss of the ability to view and copy programs.

Upload Protection outside DCS800

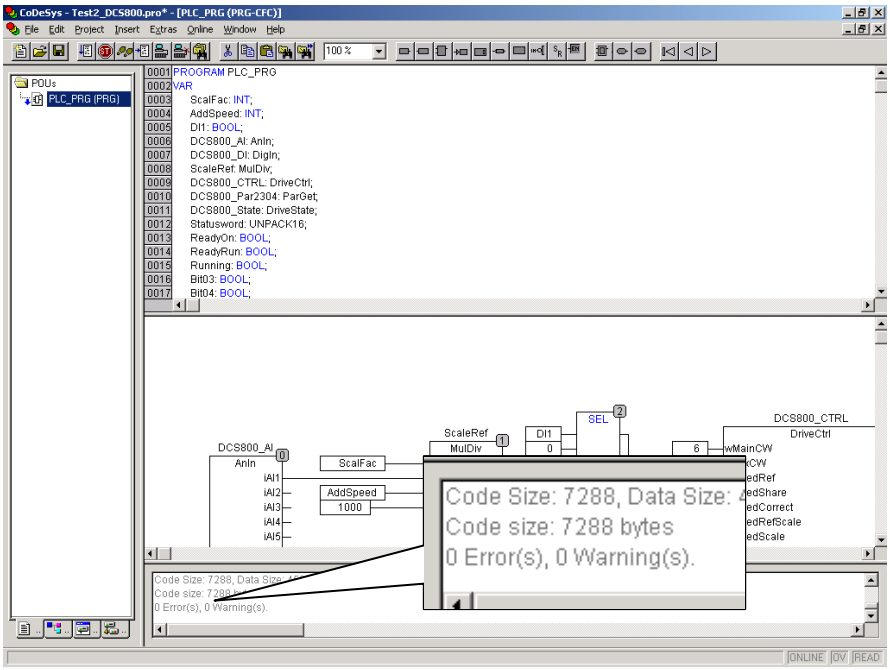
The DCS800 Drive Library contains the function block **ProgProtect**. If activated, e.g. in Init task, it is not possible to upload the application from the drive to the PC.

Build the Project

After application is completed build up the project by using **Build** (F11) in the **Project** menu.



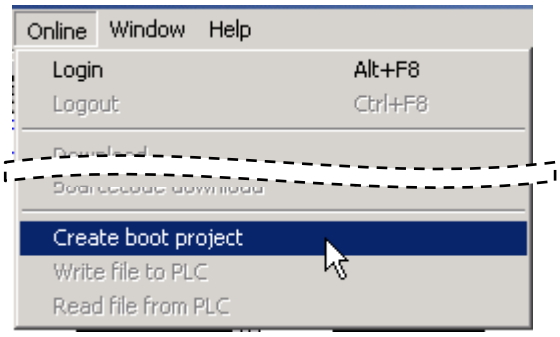
Please check in the message window that there are no errors and no warnings. All errors must be corrected before you can proceed.



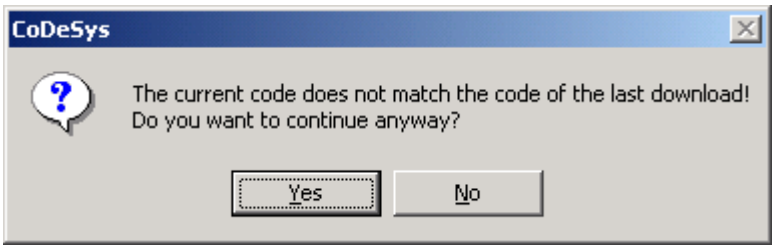
Analyse errors:
A click to an error shows the relevant line in the program.

Create Boot Files

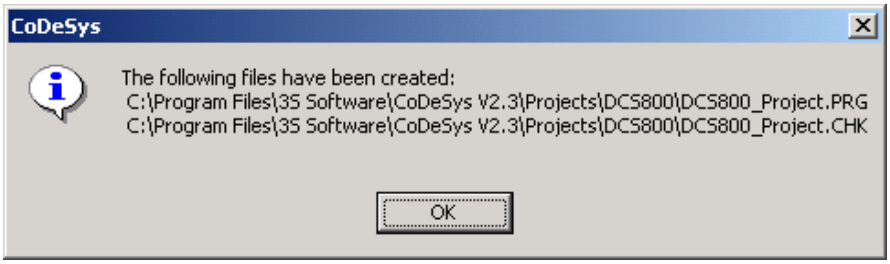
To download using DriveWindow light (DWL), it is necessary to create boot files (*.PRG and *.CHK).



While logged out, please select “**Create boot project**“ in the **Online** menu.



Accept with **Yes**.



Two files are created with the names:

<project name>**.PRG** → Boot project
<project name>**.CHK** → Checksum for boot project code

Downloading into DCS800

Chapter overview

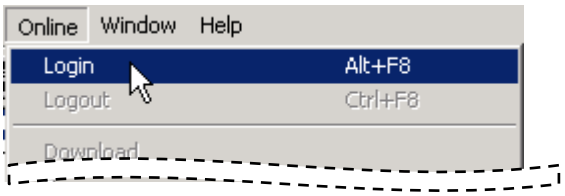
This chapter describes how to download software applications into the DCS800 either by using the CoDeSys tool or by using Drive Window light (DWL).

Before download

If writing over an existing application in the DCS800 memory card, use the function ***Clean All*** in the **project** menu to delete the old files. Otherwise there can be problems with the parameter list!
Downloading information can be found in files with the extension **SDB*, **SYM*, **RI* and **TXT*. These files do not include information about the application program. The program code can be found in the file with the extension **PRO*.

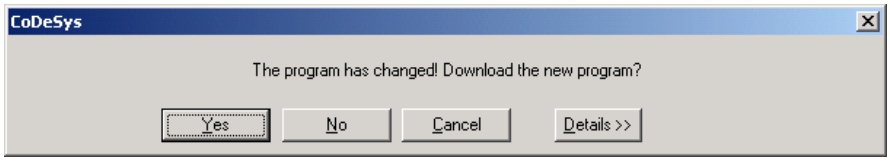
Downloading program by using CoDeSys

After the software application is built up without any errors, it can be downloaded into the DCS800.

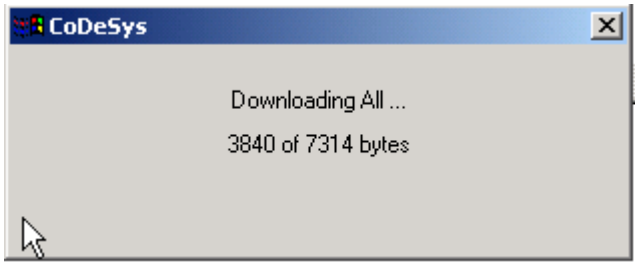


Start with **Login** in the **Online** menu.

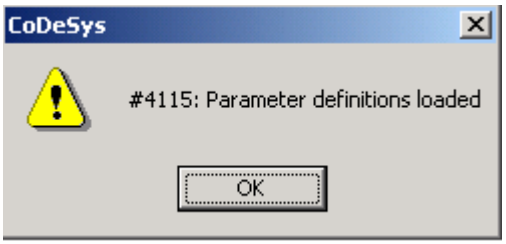
If the program has changed, the following window appears.



After pressing **Yes,** the software application will be downloaded and the tool starts downloading

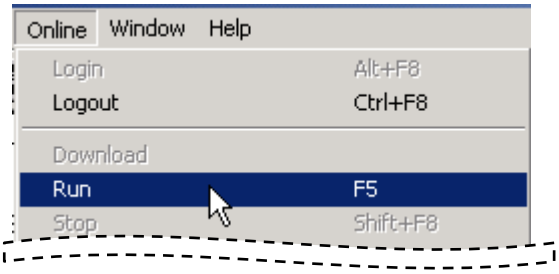


If parameters are created with CoDeSys, the following window will appear.



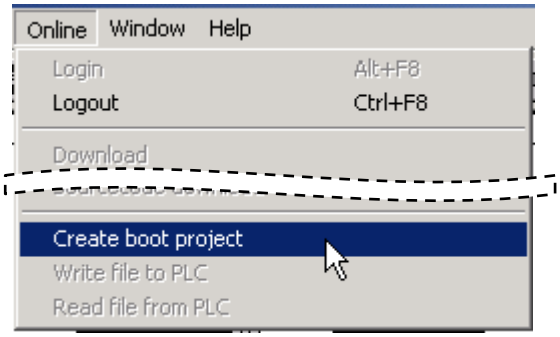
Please note that the program is still not saved.

Select **Run** in the **Online** menu to make the program begin to run.

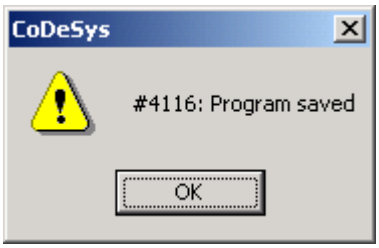


With command *Login* the code isn't saved at ABB Memory Card. It is only in the RAM.

To save into the ABB Memory Card; select "Create boot project" in the **Online** menu.

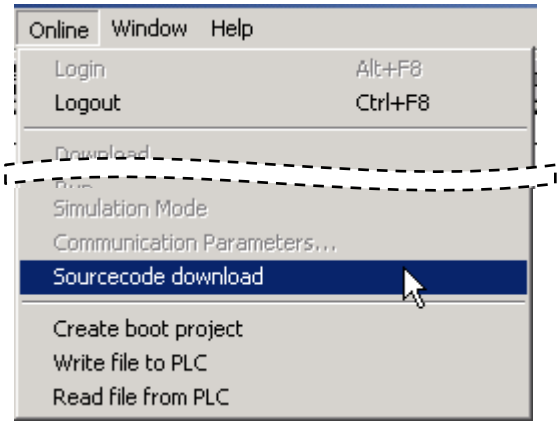


The successful result will be shown in a pop-up window.



Downloading source by using CoDeSys

If desired the source code can also be saved into ABB Memory Card as following:

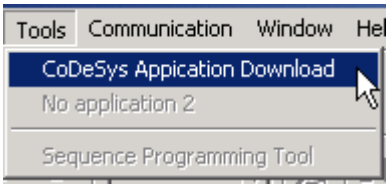


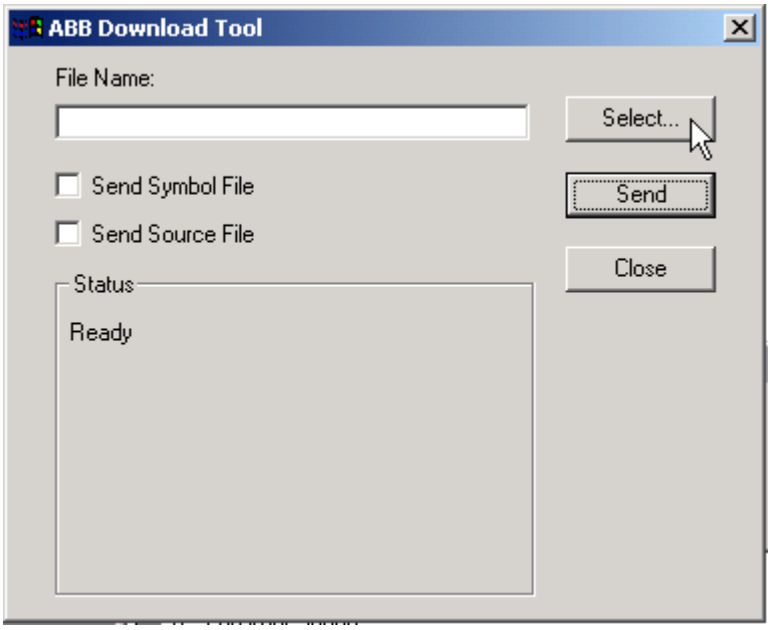
Attention:
If the source code is saved on the ABB Memory Card, it is possible to upload the program to another PC unless it is password protected! “See section “upload protection outside DCS800.”

Downloading program and source by using DWL

Connect the PC to the DCS800 and start the DriveWindow light software tool.

In Tools menu select “CoDeSys Application Download”





Select the boot project file <project>.prg, to be downloaded.

(Selection **“Send Symbol File”** is not supported yet.)

Selection **“Send Source File”** is to be chosen if the application source should also be saved to the ABB Memory Card.

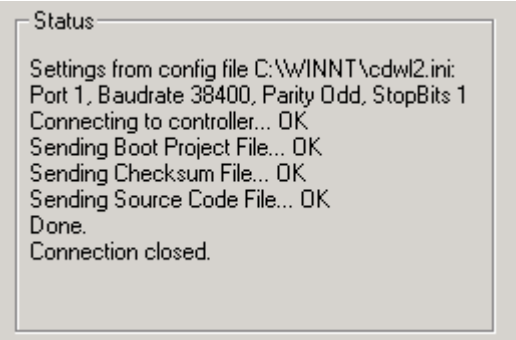
Then press **Send**.

The following files will be downloaded to ABB Memory Card:

<project>.prg (Program Code)
<project>.chk (Checksum File)

<project>.src (Source Code, optional)

During downloading the actual status will be shown.



Please note:

The filenames <project>.prg and <project>.chk will be downloaded as *DEFAULT.PRG* and *DEFAULT.CHK*.

To activate the application program, parameter P16.06 must be set:

P16.06 (ParApplSave) = EnableAppl

Upload from DCS800

Chapter overview

This chapter describes how to upload programs from the DCS800 to the PC.

General

Uploading program code is possible with the CoDeSys program only if the code is **not protected** with function block **ProgProtect**.

Uploading source code is possible with the CoDeSys program only if the source code **was previously downloaded** and if **not protected** with function block **ProgProtect**.

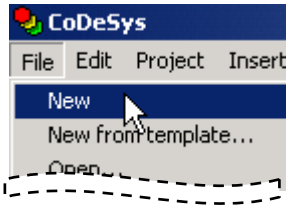
It is not possible to upload programs using Drive Window Light (DWL).

Uploading program code

Because all program code files are called *DEFAULT*, it is recommended to create a new directory before uploading.

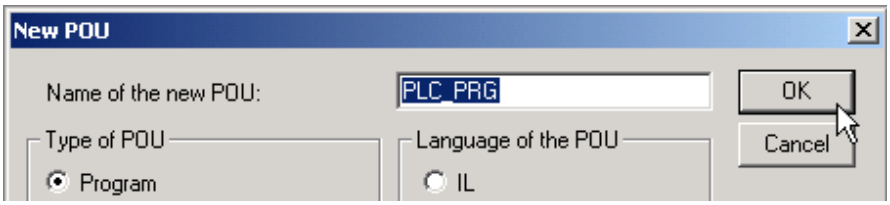
Connect the CoDeSys program with the drive.

For setting in *Login* mode please open a new project,

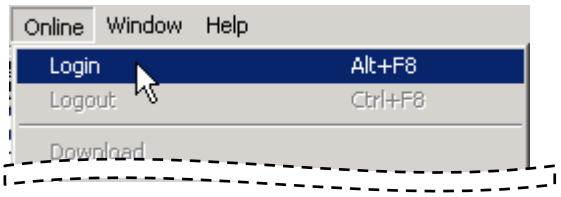


select DCS800 as target

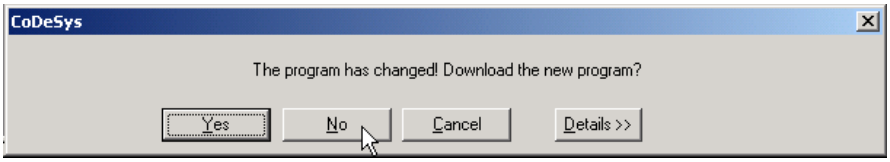
and select *OK* on *New POU* window.



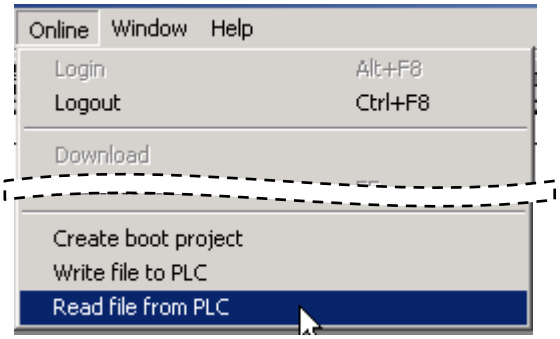
Now *Login* can be done.



The program will find out, that the program is different.



Please press only **No**!

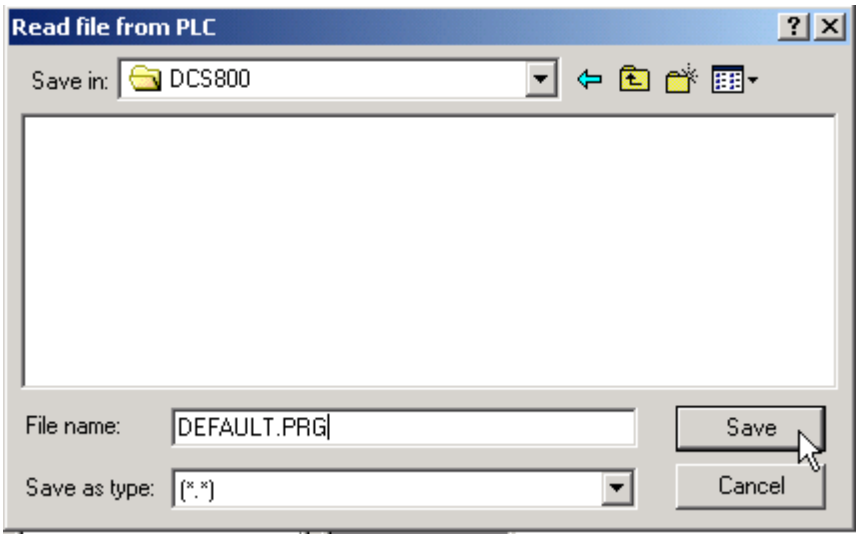


Start *Read file from PLC* of *Online* menu.

Select the new directory in *Save in*.

Write *DEFAULT.PRG* as *File name*.

Please note: The filename of program code is only *DEFAULT.PRG* in upper letters.

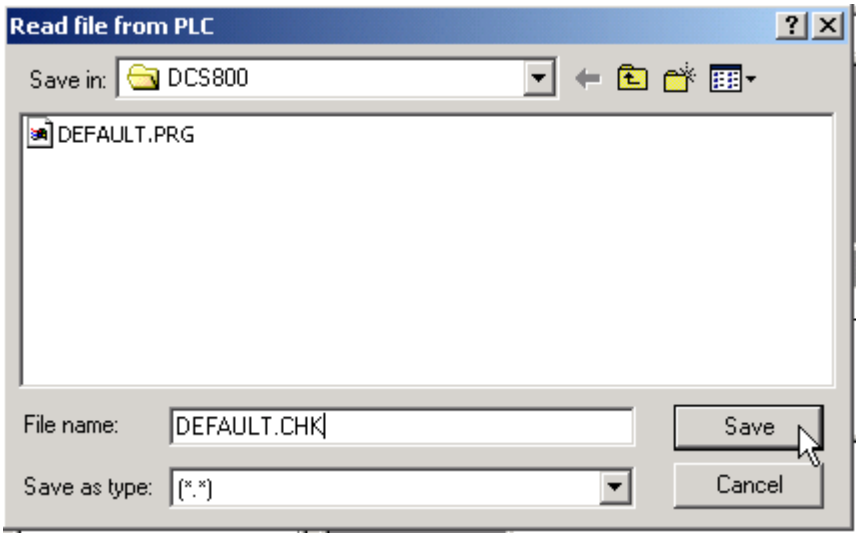


Press **Save**.

Now upload also the file of the checksum.

Write *DEFAULT.CHK* as “File name”.

Please note: The filename of program code is only *DEFAULT.CHK* in upper case letters.



Press **Save**.

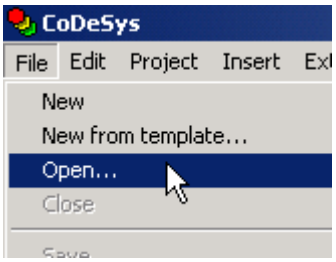
Both uploaded files should be renamed for better identification.

Both uploaded files can be used for further downloading by using the CoDeSys tool or the DriveWindow light tool.

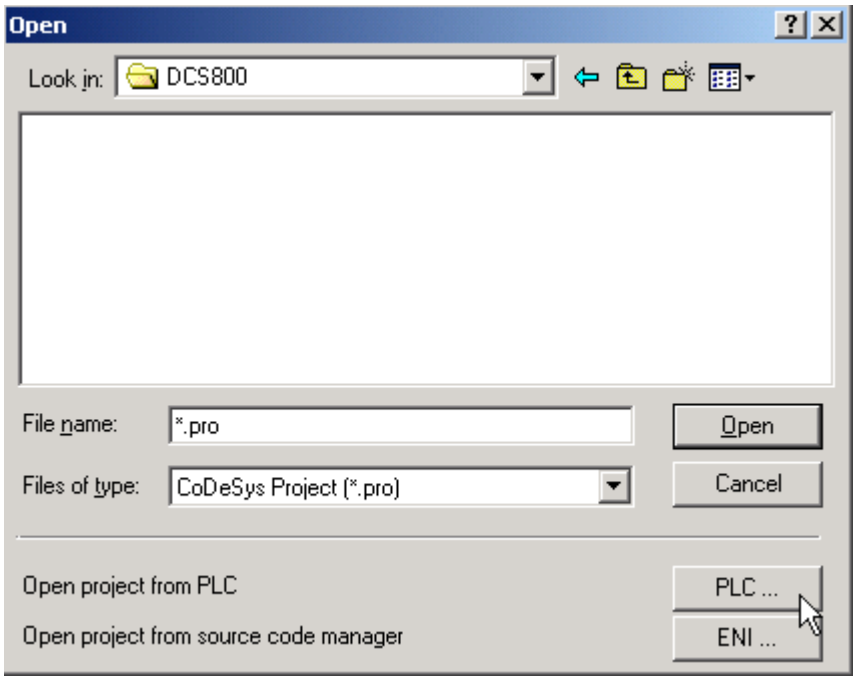
Uploading source code

If the source code of the project was saved on the ABB Memory Card, an upload of an application program is possible.

Open a project

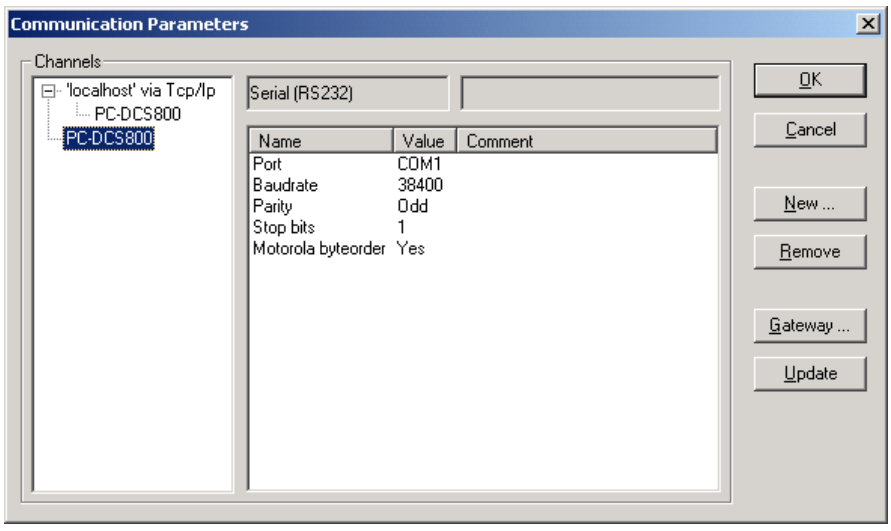


and select “Open project from PLC”



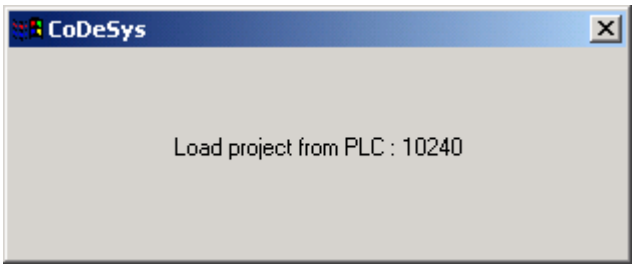
Select **DCS800** as target system…

and the communication channel.

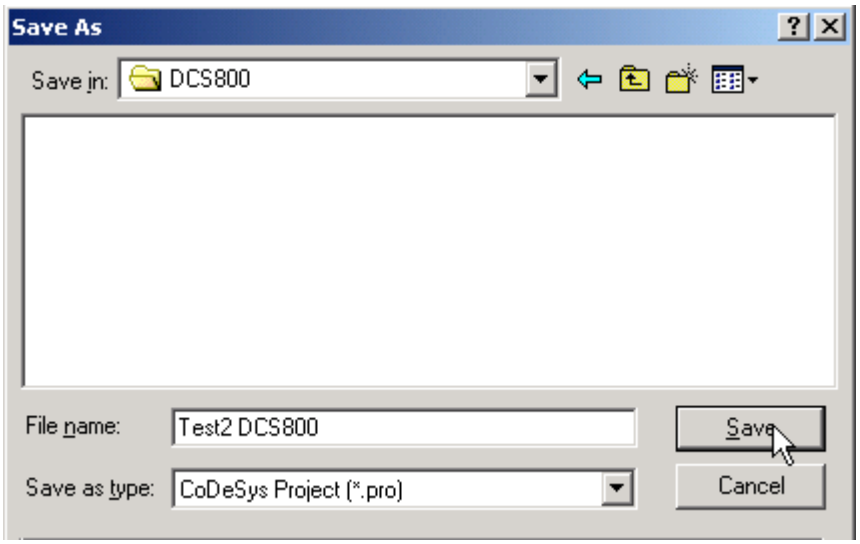


Then press **OK**.

Now the project code will be uploaded …



After loading, it can be saved.



If desired change the file name and press **Save**.

Now you can start working with this “opened” project.

Attention:
Upload of source code is only possible if the source code is present on ABB Memory Card and not password protected. If protected, please type in password to unlock!

Delete a program

Chapter overview

This chapter describes how to erase application programs. Also, the “repair mode” for ABB Memory Cards is described.

General

There are three ways to delete a program using CoDeSys:

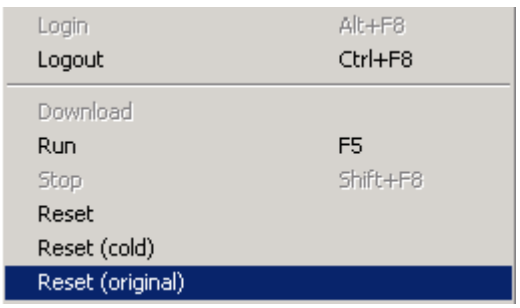
- **Reset**
  This command resets all variables to that specific value with which they were initialised.
- **Reset (cold)**
  This command sets all variables to their initialisation value. The situation is that which occurs at the start of a program which has just been downloaded to the PLC.
- **Reset (original)**
  This command resets all variables, including the remnant ones, to their initialisation values and erases the user program on the controller.

Another way to delete a program or repair an ABB Memory Card is to use parameter P16.06 **DeleteAppl**.

Functions

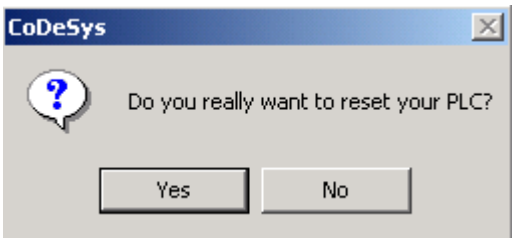
Find the functions for resetting in *Online* menu.

It is necessary to *login* the system first. (See previous section.)

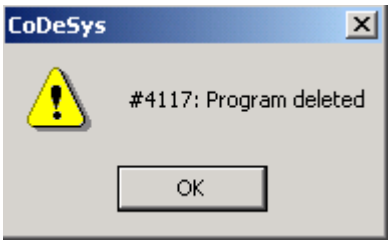


Delete a program from the Memory Card using CoDeSys

Login the system and choose **Reset (original).**



If you click **Yes**, the DCS800 Memory Card will be deleted.



When the program is deleted, you get this message.

Delete a program or repair an ABB Memory Card using firmware

Deleting of programs from the ABB Memory Card or repairing a damaged ABB Memory Card is possible with DCS800 firmware. The approach to do this is like the following:

- Set parameter P99.06 (Service Mode) to *DeleteAppl*
- Set parameter P16.06 (ParApplSave) to *DeleteAppl*

It is necessary to use this combination to protect the delete functionality because of mismatching!

Note:
If *DeleteAppl* is selected the application program and the parameter set will be deleted from the ABB Memory Card.
A damaged ABB Memory Card can be repaired using this routine. After that the application program has to be downloaded again. But only special formatted Memory Cards from ABB can be used!

Procedure to implement an application

Chapter overview

This chapter will give you an overview of how to implement an application in the DCS800.
An important tool is the so called **online-change** in CoDeSys. DCS800 supports this function in certain cases.

Create a new application

Before a new application will be activated it is necessary to check the following points:

- Needed Inputs and Outputs (analogue and digital)
- Timing of the application
- Functions and Function Blocks (e.g. Libraries)
- Signals from DCS800 firmware
- Signals to DCS800 firmware
- Interface between CoDeSys and firmware
- New parameters (group 60…69)
- New events (fault, alarm, notice)
- Documentation of the program

Enable Application

After the program is engineered and tested (e.g. by Demo Unit) it will be necessary to download the application into the DCS800. Then the application must be saved on ABB Memory Card (Compact Flash).
By activation of the application (e.g. **Run** in CoDeSys or P16.06: *Enable Application*) the program is switched-on. Command **Run** and *Enable Application* do the same.

Note:
User defined parameters (group 60 until 69) can be set to *Default* status by *Factory* macro (P99.08).

Online Change

An *Online Change* is a change of the application while the run modus. DCS800 accepts small *online changes* of the application for an online change. If the task is changed or big changes of the program are necessary an online change isn't possible. In this case a new download of the entire program is required.
If the application should be changed online the parameters keep the old value as long as there are no changes to the parameter list.

Attention:
Online changes could cause a risk to the drive system. If there is a possibility to download the entire application program it is recommended to do this.

Appendix A - Libraries

Overview about available libraries for DCS800

ABB provides several libraries with function blocks in then, but only the *DCS800 Drive Library* and the *Standard Library* will be shipped with the drive and installed automatically with CD ROM 3ADW000211. The other libraries must be ordered and installed separately.
Please contact your local ABB sales office.

Standard

The *Standard* Library (**Standard.lib**) is a hardware independent library developed by 3S-software, the manufacturer of CoDeSys software tool. It includes counter, timer and bistable functions.

DCS800 Drive Library

The *DCS800 Drive* Library (**DCS800.lib**) includes interface function blocks for communication with DCS800 firmware parameters. These blocks are required for parameter reading and writing, event handling and for digital and analogue interface communication.

Arithmetic

The *Arithmetic* Library (**DCS800Arithmetic.lib**) includes function blocks for processing with numeric values. There are an integrator, a filter and a ramp generator available. Use this library for mathematic calculations.

Special

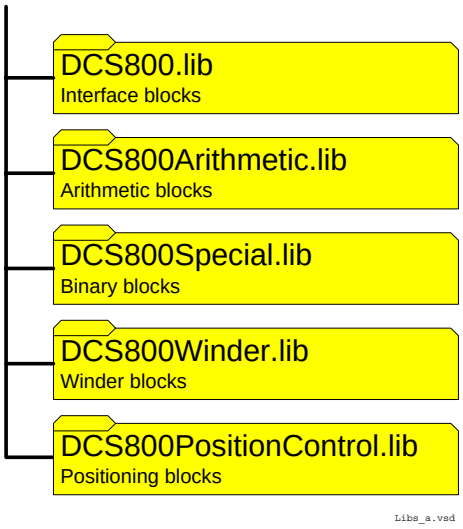
The *Special* Library (**DCS800Special.lib**) includes function blocks for binary operations and especially applications. It can be used for packing and extracting boolean values. Also a gear calculator is available.

Positioning

The *Positioning* Library (**DCS800PositionControl.lib**) includes function blocks for linear motion and synchronous motion, e.g. a position controller with an included position interpolator. There are further blocks to calculate motion control parameters and signals.
Find description of this library in manual 3ADW000348.

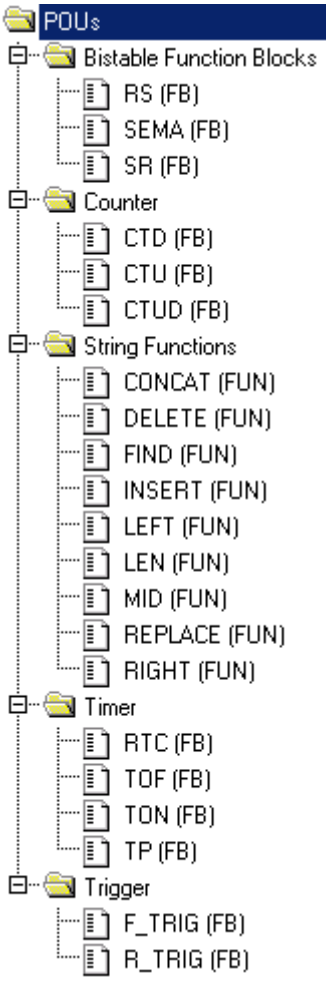
Winder

The *Winder* Library (**DCS800Winder.lib**) includes function blocks for winder applications. Several function blocks are used to calculate internal winder signals and parameters.
Description of this library is not included in this manual!


DCS800.lib
Interface blocks
DCS800Arithmetic.lib
Arithmetic blocks
DCS800Special.lib
Binary blocks
DCS800Winder.lib
Winder blocks
DCS800PositionControl.lib
Positioning blocks
Libs_a.vsd

Standard Library

Name: **Standard.lib**
Version: **all**
Firmware: **all**

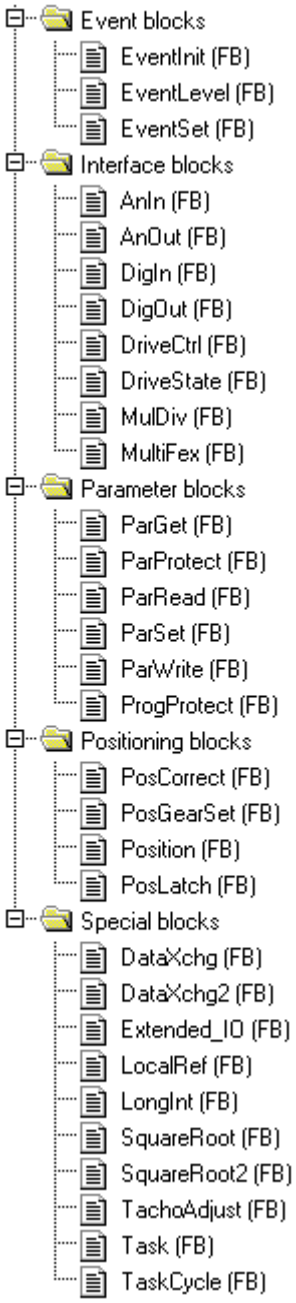


	Internal library: An internal library is programmed using IEC programming languages. The program code is available only on the PC. Function blocks can be transferred inside CoDeSys.	

A description of these blocks can be found in the CoDeSys Help-Wizard or the CoDeSys Manual!

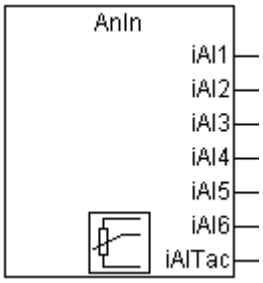
DCS800 Drive Library

Name: **dcs800lib.lib (dcs800_180.lib)**
Version: **1.80**
Firmware: **3.3 or newer**



AnIn

The function block **AnIn** can be used to read the value of all analogue inputs of the DCS800.

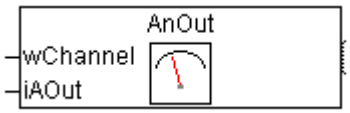


Name	LL	UL	Def	Scale	Unit	Type
iAI…	-10000	10000	--	1000 = 1V	--	INT
	Value of input … (10000 = 100% to AI scaling of DCS800 firmware)					
iAITac	-11000	11000	--	1000 = 1V	--	INT
	Value of analogue tacho (see description in firmware manual P5.01)					

Note:
AI5: AI1 of 1st analogue I/O option module
AI6: AI2 of 1st analogue I/O option module
The second analogue option module can be used only for temperature measuring!

AnOut

The function block **AnOut** can be used to write a value to an analogue output of the DCS800.

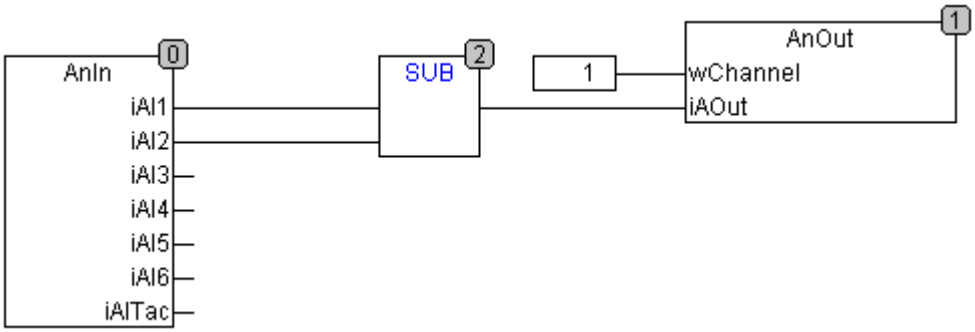


Name	LL	UL	Def	Scale	Unit	Type
wChannel			--	--	--	WORD
	Number of the analog output: 1: AO1 of DCS800 CON-4 board 2: AO2 of DCS800 CON-4 board 3: AO3 → AO1 of 1st analog I/O option module 4: AO4 → AO2 of 1st analog I/O option module					
iAOut	-32768	32767	--	10000=100 %	--	INT
	Output value					

Note:
The result of **AnOut** will be written to one of the *CtrlWord* parameters in group 15. For connection with an analogue output of the drive, please write the *CtrlWord* parameter number in the corresponding *index* in parameter group 15. (see example; see firmware manual)

Example:
Two values from analogue input 1 (AI1) and 2 (AI2) should be subtracted from each other

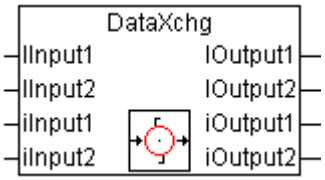
and written to analogue output 1 (AO1).



Function block **AnOut** writes the value *iAOut* to parameter P15.02 of DCS800 firmware. To connect this value with the physical analogue output of the drive, please connect *CtrlWordAO1* to the index of the related parameter (in this case: P15.01: **1502**).

DataXchg

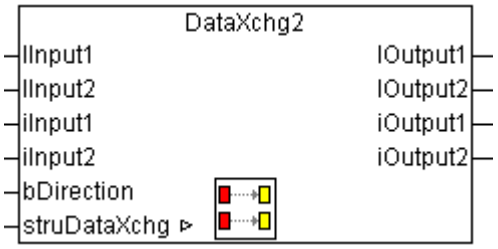
Function block **DataXchg** (data exchange) can be used to transfer input values to the outputs without any interrupts. It confirms a consistent data exchange.



Name	LL	UL	Def	Scale	Unit	Type
lInput1	-2147483648 Input value 1	2147483647	--	--	--	DINT
lInput2	-2147483648 Input value 2	2147483647	--	--	--	DINT
iInput1	-32768 Input value 1	32767	--	--	--	INT
iInput2	-32768 Input value 2	32767	--	--	--	INT
lOutput1	-2147483648 Output value 1	2147483647	--	--	--	DINT
lOutput2	-2147483648 Output value 2	2147483647	--	--	--	DINT
iOutput1	-32768 Output value 1	32767	--	--	--	INT
iOutput2	-32768 Output value 2	32767	--	--	--	INT

DataXchg2

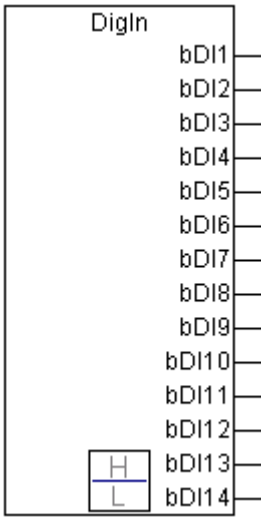
Function block **DataXchg2** (data exchange) can be used to transfer input values to the outputs without any interrupts. It confirms a consistent data exchange.
Difference to **DataXchg** is the possibility to read or write to global variables.



Name	LL	UL	Def	Scale	Unit	Type
lInput1	-2147483648 Input value 1	2147483647	--	--	--	DINT
lInput2	-2147483648 Input value 2	2147483647	--	--	--	DINT
iInput1	-32768 Input value 1	32767	--	--	--	INT
iInput2	-32768 Input value 2	32767	--	--	--	INT
bDirection	FALSE FALSE: Read from global variables TRUE: Write to global variables	TRUE	FALSE	--	--	BOOL
stru-DataXchg	Exchange of a data structure without interrupts					STRUCT
lOutput1	-2147483648 Output value 1	2147483647	--	--	--	DINT
lOutput2	-2147483648 Output value 2	2147483647	--	--	--	DINT
iOutput1	-32768 Output value 1	32767	--	--	--	INT
iOutput2	-32768 Output value 2	32767	--	--	--	INT

DigIn

Function block **DigIn** is used to read the status of all digital inputs (P8.05 DIStatusWord)



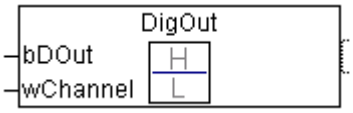
Name	LL	UL	Def	Scale	Unit	Type
bDI...	FALSE	TRUE	FALSE	--	--	BOOL
	State of digital input…					

Note:
DI1…8: DI's of DCS800 CON-4 board
DI9…11: DI's of 1st extension module
DI12…14: DI's of 2nd extension module

Attention:
Inverted parameters of system software are not applied!

DigOut

Function block **DigOut** is used to write a value to a digital output.



Name	LL	UL	Def	Scale	Unit	Type
wChannel	1 Number of digital output	12	1	--	--	WORD
bDOut	FALSE Output value	TRUE	FALSE	--	--	BOOL

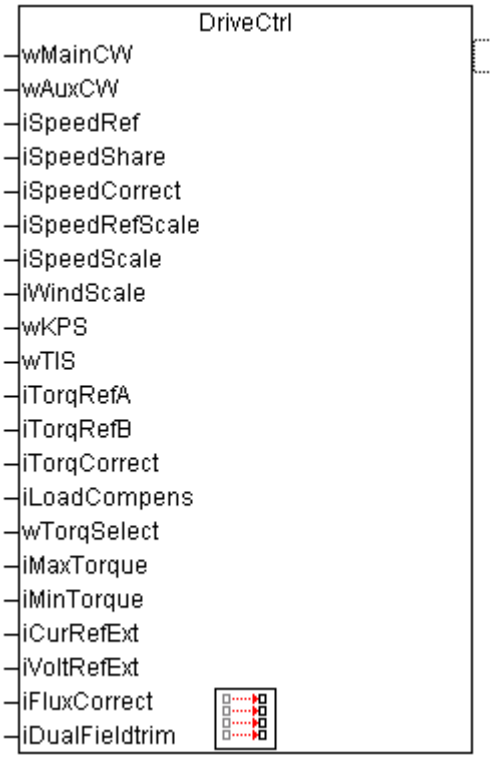
Note:

DO	1…8:	DO's of DCS800 CON-4 board
DO	9…10:	DO's of 1st extension module
DO	11…12:	DO's of 2nd extension module

Function block **DigOut** writes the value *bDOut* to parameter P7.05 of DCS800 firmware.
To connect this value with the physical digital output of the drive, please connect *DOCtrlWord* to the index and bit of the related parameter of group 14.

DriveCtrl

Function block **DriveCtrl** can be used to write values to drive control parameters.

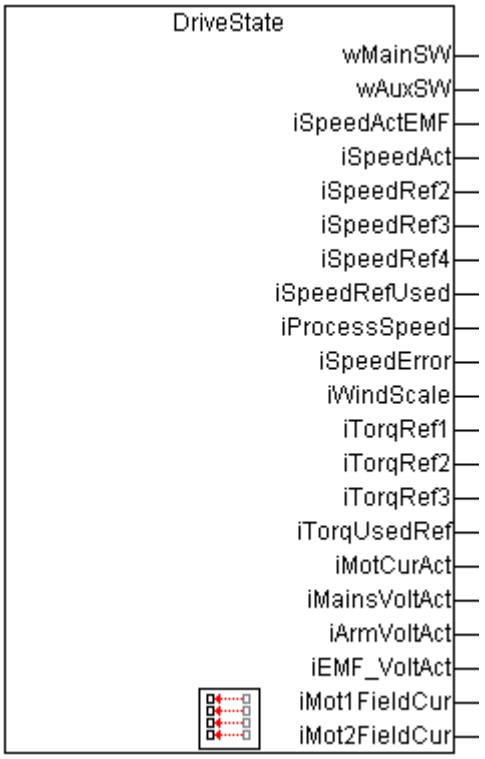


Note:
All unconnected inputs will be written to the correspondent destination parameter which means that the value 0 will be written. This cause problems!
The destination parameter will not be written with 0 if the open input is connected to a constant of 100 000.

Name	LL	UL	Def	Scale	Unit	Type
wMainCW	0 MainCtrlWord (7.01)	65535	100000	--	--	WORD
wAuxCW	0 AuxCtrlWord (7.02)	65535	100000	--	--	WORD
iSpeedRef	-32768 SpeedRef (23.01)	32767	100000	--	--	INT
iSpeedShare	-32768 Speed Share (23.05)	32767	100000	--	--	INT
iSpeedCorr	-32768 SpeedCorr (23.04)	32767	100000	--	--	INT
iSpeedRefSc	-32768 SpeedRefScale (23.16)	32767	100000	--	--	INT
iSpeedScale	-32768 M1 SpeedScale (50.01)	32767	100000	--	--	INT
iWindScale	-32768 WinderScale (50.17)	32767	100000	--	--	INT
wKPS	0 KpS (24.03)	65535	100000	--	--	WORD
wTIS	0 TiS (24.09)	65535	100000	--	--	WORD
iTorqRefA	-32768 TorqueRef A (25.01)	32767	100000	--	--	INT
iTorqRefB	-32768 TorqueRef B (25.04)	32767	100000	--	--	INT
iTorqCorrect	-32768 TorqueCorrect (26.15)	32767	100000	--	--	INT
iLoadCompe	-32768 LoadComp (26.02)	32767	100000	--	--	INT
wTorqSelect	0 TorqSel (26.01)	65535	100000	--	--	WORD
iMaxTorque	-32768 TorqueMax (20.05)	32767	100000	--	--	INT
iMinTorque	-32768 TorqueMin (20.06)	32767	100000	--	--	INT
iCurRefExt	-32768 CurRefExt (43.03)	32767	100000	--	--	INT
iVoltRefExt	-32768 VoltRefExt (46.02)	32767	100000	--	--	INT
iFluxCorrect	-32768 FluxCorr (46.09)	32767	100000	--	--	INT
iDualField-trim	-32768 FldCurTrim (45.17)	32767	100000	--	--	INT

DriveState

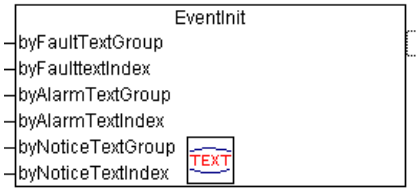
Function block **DriveState** can be used to read values from actual drive signals. These are the most significant actual parameters.



Name	LL	UL	Def	Scale	Unit	Type
wMainSW	0	65535	--	--	--	WORD
	MainStatWord (8.01)					
wAuxSW	0	65535	--	--	--	WORD
	AuxStatWord (8.02)					
iSpeedActEMF	-32768	32767	--	--	--	INT
	SpeedActEMF (1.02)					
iSpeedRef2	-32768	32767	--	--	--	INT
	SpeedRef 2 (2.01)					
iSpeedRef3	-32768	32767	--	--	--	INT
	SpeedRef 3 (2.02)					
iSpeedRef4	-32768	32767	--	--	--	INT
	SpeedRef 4 (2.18)					
iSpeedRe-fUsed	-32768	32767	--	--	--	INT
	SpeedRefUsed (2.17)					
iProcessSpeed	-32768	32767	--	--	--	INT
	Process Speed (1.41)					
iSpeedError	-32768	32767	--	--	--	INT
	Speed Error (2.03)					
iWindScale	-32768	32767	--	--	--	INT
	Speed Scale Act (2.29)					
iTorqRef1	-32768	32767	--	--	--	INT
	TorqueRef 1 (2.08)					
iTorqRef2	-32768	32767	--	--	--	INT
	TorqueRef 2 (2.09)					
iTorqRef3	-32768	32767	--	--	--	INT
	TorqueRef 3 (2.10)					
iTorqUsedRef	-32768	32767	--	--	--	INT
	Torque Ref Used (2.13)					
iMotCurAct	-32768	32767	--	--	--	INT
	MotCur (1.06)					
iMainsVoltAct	-32768	32767	--	--	--	INT
	MainsVoltActRel (1.11)					
iArmVoltAct	-32768	32767	--	--	--	INT
	Arm Volt Act (1.13)					
iEMF_VoltAct	-32768	32767	--	--	--	INT
	EMF Volt Act Rel (1.17)					
iMot1FieldCur	-32768	32767	--	--	--	INT
	Mot1FieldCurRel (1.29)					
iMot2FieldCur	-32768	32767	--	--	--	INT
	Mot2FieldCurRel (1.31)					

EventInit

Function block **EventInit** can be used to allocate event texts to user events.

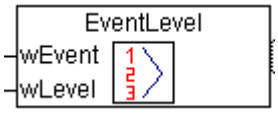


Name	LL	UL	Def	Scale	Unit	Type
byFaultTextGroup	60 Fault Text Group	69	--	--	--	BYTE
byFaultTextIndex	1 Fault Text Index	99	--	--	--	BYTE
byAlarmTextGroup	60 Alarm Text Group	69	--	--	--	BYTE
byAlarmTextIndex	1 Alarm Text Index	99	--	--	--	BYTE
byNoticeTextGroup	60 Notice Text Group	69	--	--	--	BYTE
byNoticeTextIndex	1 Notice Text Index	99	--	--	--	BYTE

Note:
See example of function block **EventSet** to understand the functionality!

EventLevel

Function block **EventLevel** can be used to define the alarm or fault level for user events.



Name	LL	UL	Def	Scale	Unit	Type
wEvent	Event number: 310…325: Alarms 610…625: Faults		0	--	--	WORD
wLevel	Event level: 1…4: Alarm 1…5: Fault		0	--	--	WORD

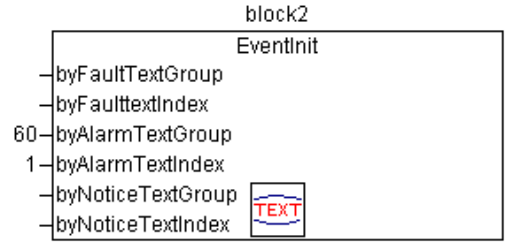
Note:
Please find the description about alarm and fault levels in the DCS800 firmware manual!

EventSet

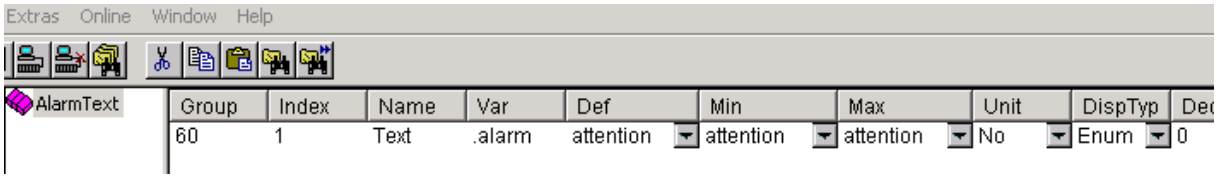
Function block **EventSet** is used to activate user faults, alarms and notices. The second function is to set system faults and alarms or deactivate them.

Name	LL	UL	Def	Scale	Unit	Type
bActivate	FALSE * Set an event	TRUE*	FALSE	--	--	BOOL
iEvent	<u>Set or activate an event:</u> 101…199: System Alarm 310…325: User Alarm 501…599: System Fault 610…625: User Fault / Trip 810…825: User Notice <u>Disable or deactivate an event:</u> **10**101…**10**199: System Alarm **(alarm number + 10000)** **10**501…**10**599: System Fault **(fault number + 10000)** *A negative fault number trips the DC Breaker in addition to the fault reaction!*			--	--	INT
iDelay	0 Event is delayed by the time defined at this input.	30000	0	--	ms	INT
bState	FALSE FALSE: event not active, TRUE: event activated	TRUE	FALSE	--	--	BOOL

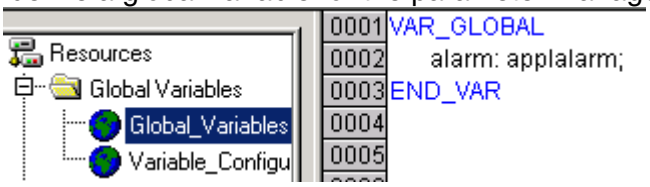
<u>Example 1 (User Event):</u>
In this example an alarm message will be activated with a Boolean value (variable Active). With the function block **EventInit** there is the possibility to allocate an alarm text, which is saved in parameter 60.01. The text which is declared as type ENUM can be selected in the parameter manager columns Def, Min and Max.



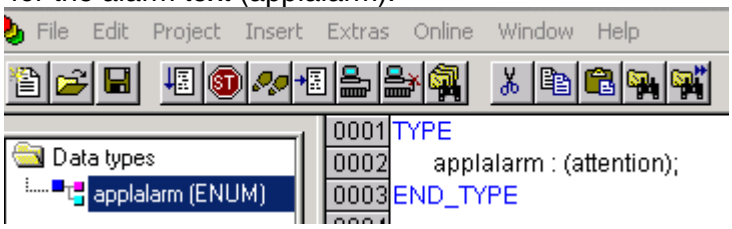
Now it is necessary to define a parameter for the text in the parameter manager:



It is also necessary to define a global variable for the parameter manager:



Create a data type for the alarm text (applalarm):



The alarm text is in brackets. Define more texts with a comma between the texts, e.g. (attention1, attention2, attention3).

If you activate the alarm (variable Active = TRUE), after 1000 ms the alarm switches on and you will see the message text, e.g. on the panel.

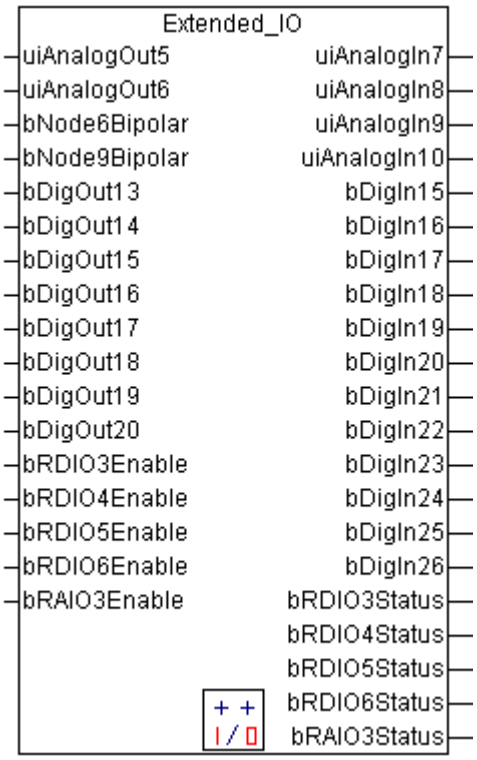
Example 2 (System Event):

In this example the system event (F532 Motor Over speed) should be set with function block **EventSet**. Write the fault number 532 to input *iEvent* and activate the fault with input *bActivate*.

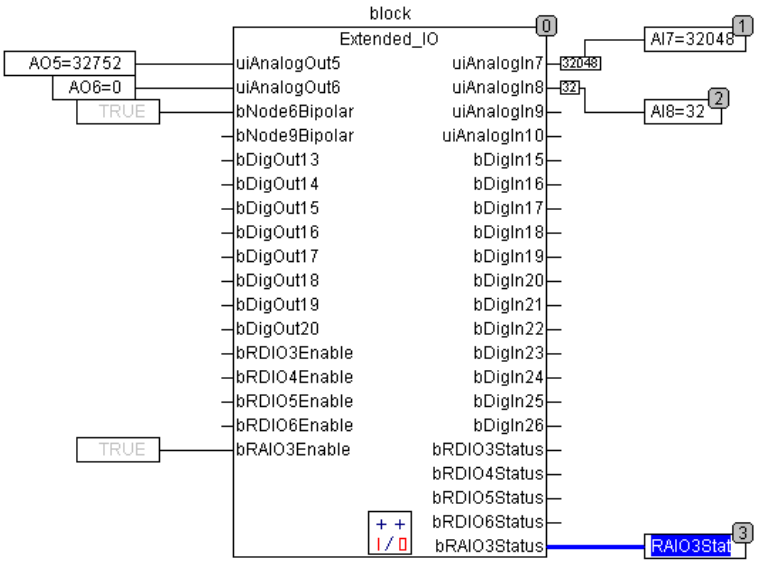
Note:
Faults must be reset with system reset command. An alarm stops if input *bActivate* is FALSE.

Extended_IO

Function block **Extended_IO** is used to expand the amount of analogue inputs and outputs as well as digital inputs and outputs.



Example for RAIO 3:



uiAnalogIn7 detects (circa): +10 V == $7FF0_{Hex}$ == 32752_{Dec}
uiAnalogOut5 gets (circa): 32752_{Dec} == $7FF0_{Hex}$ == 10 mA
bNode6Bipolar == TRUE > Scaling of analog values is bipolar!

Name	LL	UL	Def	Scale	Unit	Type
uiAnalogOut5	0	65535	--	--	--	WORD
	1st analogue output of RAIO adapter with DDCS node number 6					
uiAnalogOut6	0	65535	--	--	--	WORD
	2nd analog output of RAIO adapter with DDCS node number 6					
bNode6Bipolar	FALSE	TRUE *	FALSE	--	--	BOOL
	* Bipolar scaling of RAIO 3					
bNode9Bipolar	FALSE	TRUE *	FALSE	--	--	BOOL
	* Bipolar scaling of RAIO 2 (RAIO for temperature measurement!!!)					
bDigOut13	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital output of RDIO adapter with DDCS node number 11					
bDigOut14	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital output of RDIO adapter with DDCS node number 11					
bDigOut15	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital output of RDIO adapter with DDCS node number 12					
bDigOut16	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital output of RDIO adapter with DDCS node number 12					
bDigOut17	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital output of RDIO adapter with DDCS node number 13					
bDigOut18	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital output of RDIO adapter with DDCS node number 13					
bDigOut19	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital output of RDIO adapter with DDCS node number 14					
bDigOut20	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital output of RDIO adapter with DDCS node number 14					
bRDIO3Enable	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable RDIO module with DDCS node number 11					
bRDIO4Enable	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable RDIO module with DDCS node number 12					
bRDIO5Enable	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable RDIO module with DDCS node number 13					
bRDIO6Enable	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable RDIO module with DDCS node number 14					
bRAIO3Enable	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable RAIO module with DDCS node number 6					
uiAnalogIn7	0	65535	--	--	--	WORD
	1st analog input of RAIO adapter with DDCS node number 6					
uiAnalogIn8	0	65535	--	--	--	WORD
	2nd analog input of RAIO adapter with DDCS node number 6					
uiAnalogIn9	0	65535	--	--	--	WORD
	1st analog input of RAIO adapter for motor temperature measurement / DDCS node number 9					
uiAnalogIn10	0	65535	--	--	--	WORD
	2nd analog input of RAIO adapter for motor temperature measurement / DDCS node number 9					
bDigIn15	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital input of RDIO adapter with DDCS node number 11					
bDigIn16	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital input of RDIO adapter with DDCS node number 11					
bDigIn17	FALSE	TRUE	FALSE	--	--	BOOL
	3rd digital input of RDIO adapter with DDCS node number 11					

Name	LL	UL	Def	Scale	Unit	Type
bDigIn18	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital input of RDIO adapter with DDCS node number 12					
bDigIn19	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital input of RDIO adapter with DDCS node number 12					
bDigIn20	FALSE	TRUE	FALSE	--	--	BOOL
	3rd digital input of RDIO adapter with DDCS node number 12					
bDigIn21	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital input of RDIO adapter with DDCS node number 13					
bDigIn22	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital input of RDIO adapter with DDCS node number 13					
bDigIn23	FALSE	TRUE	FALSE	--	--	BOOL
	3rd digital input of RDIO adapter with DDCS node number 13					
bDigIn24	FALSE	TRUE	FALSE	--	--	BOOL
	1st digital input of RDIO adapter with DDCS node number 14					
bDigIn25	FALSE	TRUE	FALSE	--	--	BOOL
	2nd digital input of RDIO adapter with DDCS node number 14					
bDigIn26	FALSE	TRUE	FALSE	--	--	BOOL
	3rd digital input of RDIO adapter with DDCS node number 14					
bRDIO3Status	FALSE	TRUE	FALSE	--	--	BOOL
	* RDIO module with DDCS node number 11 detected					
bRDIO4Status	FALSE	TRUE	FALSE	--	--	BOOL
	* RDIO module with DDCS node number 12 detected					
bRDIO5Status	FALSE	TRUE	FALSE	--	--	BOOL
	* RDIO module with DDCS node number 13 detected					
bRDIO6Status	FALSE	TRUE	FALSE	--	--	BOOL
	* RDIO module with DDCS node number 14 detected					
bRAIO3Status	FALSE	TRUE	FALSE	--	--	BOOL
	* RAIO module with DDCS node number 6 detected					

Attention:
If none of the RDIO adapters with DDCS node numbers 12 ... 14 are enabled (*bRDIO4Enable* … *bRDIO6Enable*), the digital input signals as well as the module status signal of these adapters are forced to zero regardless of the existence of the corresponding adapter.
The analog input signals *uiAnalogIn9* and *uiAnalogIn10* are available only, if the RAIO adapter for motor temperature measurement is located on an AIMA-01 adapter connected to the SDCS-COM-8x. The analog outputs of this adapter are not supported by this function block. It must not be used for motor temperature measurement at the same time.

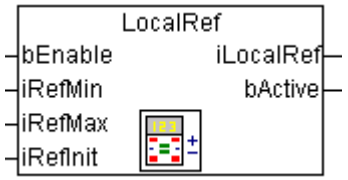
Note:
RDIO 3 … 6 and RAIO 3 have to be connected to AIMA-01 board. This board must be connected with COM-8x board via channel 1. (see DCS800 Hardware Manual)

The following node addresses must be selected:

- RDIO 3 node id 11
- RDIO 4 node id 12
- RDIO 5 node id 13
- RDIO 6 node id 14
- RAIO 3 node id 6

LocalRef

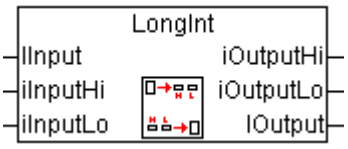
Function block **LocalRef** is used to read a value which is set by the DCS800 panel. It can be used to do a tacho adjustment in combination with function block **TachoAdjust**.



Description on request

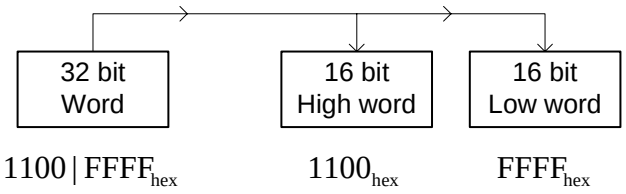
LongInt

Function block **LongInt** is used to convert between data types. It split or join between 32 bit and 16 bit types.

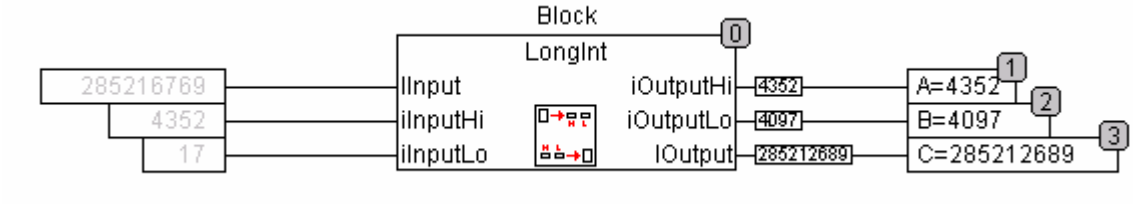


Name	LL	UL	Def	Scale	Unit	Type
lInput	0	4294967295	--	--	--	DWORD
	Input value 32 bit format					
iInputHi	0	65535	--	--	--	WORD
	Input high word 16 bit format					
iInputLo	0	65535	--	--	--	WORD
	Input low word 16 bit format					
iOutputHi	0	65535	--	--	--	WORD
	Output high word 16 bit format					
iOutputLo	0	65535	--	--	--	WORD
	Output low word 16 bit format					
lOutput	0	4294967295	--	--	--	DWORD
	Output value 32 bit format					

Description:

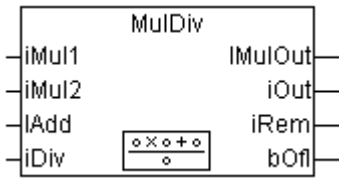


Example:



MulDiv

Function block **MulDiv** is used for safety calculations. This function block is overflow protected!
It is also runtime optimized and calculates the remain data of the division.



$$iOut = \frac{iMul1 \cdot iMul2 + lAdd}{iDiv}$$

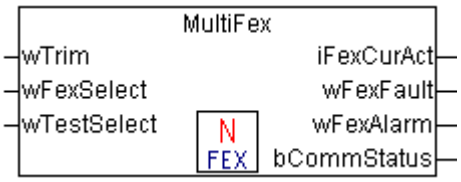
$$lMulOut = iMul1 \cdot iMul2 \quad \text{(32 bit type)}$$

Name	LL	UL	Def	Scale	Unit	Type
iMul1	-32768	32767	--	--	-	INT
	Input 1 for multiplication					
iMul2	-32768	32767	--	--	-	INT
	Input 2 for multiplication					
lAdd	-2147483648	2147483647	--	--	--	DINT
	Input to add a value to the result of the multiplication					
iDiv	-32768	32767	--	--	-	INT
	Input to divide the result of the multiplication and addition					
lMulOut	-2147483648	2147483647	--	--	--	DINT
	Output of the result of the multiplication					
iOut	-32767	32767	--	--	-	INT
	Output of the complete operation					
iRem	-32767	32767	--	--	-	INT
	Remain data of division					
bOfl	FALSE	TRUE*	FALSE	--	--	BOOL
	* Overflow occurred					

Note:
The output *iOut* is limited to – 32767…+ 32767. It works internal with 32 bit values.

MultiFex

Function block **MultiFex** is used for applications with several field exciters connected to one armature converter.
A typical application is a roller table. For each field exciter one function block **MultiFex** is needed!

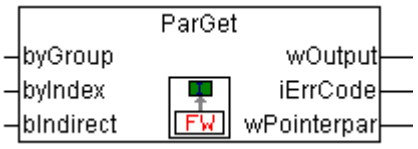


Name	LL	UL	Def	Scale	Unit	Type
wTrim	0	20000	10000	10000 == 100%		WORD
	Trimming factor for field current of selected field exciter					
wFexSelect	1	32	--	--	--	WORD
	Node number of field exciter (unique P94.08)					
wTestSelect	1	32	--	--	--	WORD
	Test routine: LED on field exciter blinks if *wFexSelect* == *wTestSelect.*					
iFexCurAct	0	20000	--	10000 == 100%		WORD
	Actual field current of selected field exciter					
wFexFault	0	65535	--	--	--	WORD
	Field exciter fault word					
wFexAlarm	0	65535	--	--	--	WORD
	Field exciter alarm word					
bCommStatus	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Field exciter communication works properly					

Note:
For more information please read manual “Multi Fex Application” 3ADW000309Rxx.

ParGet

Function block **ParGet** is used for cyclic reading from a firmware parameter.

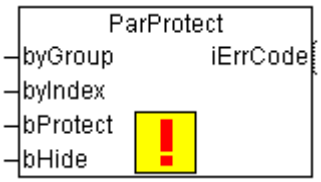


Name	LL	UL	Def	Scale	Unit	Type
byGroup	1	99	--	--	--	BYTE
	Involved parameter group					
byIndex	1	99	--	--	--	BYTE
	Involved parameter index					
bIndirect	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: byGroup and byIndex address a pointer parameter which addresses the involved parameter FALSE: byGroup and byIndex address the involved parameter direct					
wOutput	0	65535	--	--	--	WORD
	Read value					
iErrCode			--	--	--	INT
	0: o.k. 3: invalid group or index 4: selected index not allowed 5: no such group 8: no such index -…: the error code isn´t related to pointer parameter, but to the index addressed by this parameter					
wPointerpar	0	65535	--	--	--	WORD
	Group* 100 + index					

Note:
ParGet and **ParSet** are based on the internal scaling of parameters; speed scaling = 20000, torque- and current-scaling = 10000.
These blocks do not store their values into the FLASH while writing to the firmware parameters.
Not all parameters can be read with this function block! With problems on the output, look to output *iErrCode*.

ParProtect

Function block **ParProtect** can be used to protect parameters against changes or hide parameters.

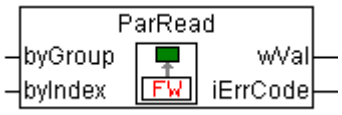


Name	LL	UL	Def	Scale	Unit	Type
byGroup	1	99	--	--	--	BYTE
	Involved parameter group					
byIndex	0	99	--	--	--	BYTE
	Involved parameter Index, if 0 applies for entire group					
bProtect	FALSE	TRUE*	FALSE	--	--	BOOL
	* Parameter or group will be write protected					
bHide	FALSE	TRUE*	FALSE	--	--	BOOL
	Parameter or group will be hidden					
iErrCode	0	8	--	--	--	INT
	0: o.k. 3: invalid group or index 5: no such group 8: no such index					

Note:
If input index is connected to the constant value 0, the entire group will be protected or hidden.

ParRead

Function block **ParRead** is used to read a value from a firmware parameter.

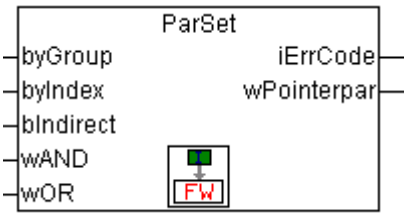


Name	LL	UL	Def	Scale	Unit	Type
byGroup	1	99	--	--	--	BYTE
	Involved parameter group					
byIndex	1	99	--	--	--	BYTE
	Involved parameter index					
wVal	0	65535	--	--	--	WORD
	Read value					
iErrCode	0	10	--	--	--	INT
	0: o.k. 3: invalid group or index 5: no such group 8: no such index					

Note:
Function blocks **ParRead** and **ParWrite** provide the same scaling as the values displayed on the panel or software tool. This means it is always the same as the physical value if existing! These blocks don't store their values into the FLASH while writing to the firmware parameters.

ParSet

Function block **ParSet** is used to write a value to a cyclic parameter. This provides the fastest connection to firmware.



Name	LL	UL	Def	Scale	Unit	Type
byGroup	1	99	--	--	--	BYTE
	Involved parameter group					
byIndex	1	99	--	--	--	BYTE
	Involved parameter index					
bIndirect	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: *byGroup* and *byIndex* address a pointer parameter, which addresses the involved parameter FALSE: *byGroup* and *byIndex* address the involved parameter direct					
wAND	-32768	32767	0	--	--	WORD
	Destination value = (destination value & *wAND*) | *wOR* **(mask input!)**					
wOR	-32768	32767	--	--	--	WORD
	Destination value = (destination value & wAND) | wOR **(normal input!)**					
iErrCode			--	--	--	INT
	0: o.k. 3: invalid group or index 4: selected index not allowed for ParSet 5: no such group 8: no such index -…: the error code isn´t related to pointer parameter, but to the index addressed by this parameter					
wPointerpar	0	65535	--	--	--	WORD
	Value of pointer					

Note:
ParGet and **ParSet** are based on the internal scaling of parameters; speed scaling = 20000, torque- and current-scaling = 10000.
These blocks don't store their values into the FLASH while writing to the firmware parameters.
If you don´t need a masked input, set input *wAND* to 0!!!

It's not possible to write to all DCS800 parameters. To debug see the output *iErrCode* of the function block!

Attention:
Writing to signals (not writable!) generate *F548 FW failure*. Diagnosis 9.11 shows the relevant parameter. In this case switch off electronics and switch-on once again!

Example:
The *Main Control Word* (7.01) should be set with the function block **ParSet**, but bits 4 through 15 must not be influenced by the CoDeSys application.

In this case it is possible to use the input *wAND* to mask bits 4 through 15 (in Hex-Format FFF0). The other bits will be set with the input *wOR*.

Function block **ParSet** works in execution order like the table below:

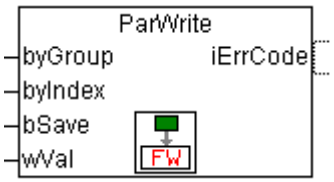
	Action		**Hex - Value**			
1.	Read P7.01	e.g.	0	4	7	X
2.	Mask P7.01	*wAND* =	F	F	F	0
3.	Value from CoDeSys application	wOR =	1	2	3	F
4.	Write to P7.01	P7.01 =	0	4	7	F

X: don't care

Explanation:
First the actual value from P7.01 has to be read (e.g. 470_{Hex}). Then bits 4 through 15 are masked, meaning only bit 0 to 3 can be written from the CoDeSys application. After that the value coming from the CoDeSys application will be written to P7.01 but only bit 0 through 3 cause a change in P7.01. Result is the value $47F_{Hex}$ in P7.01.

ParWrite

Function block **ParWrite** can be used to write to a firmware parameter.



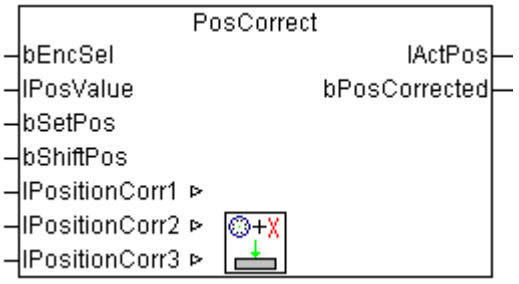
Name	LL	UL	Def	Scale	Unit	Type
byGroup	1	99	--	--	--	BYTE
	Involved parameter group					
byIndex	1	99	--	--	--	BYTE
	Involved parameter index					
bSave	FALSE	TRUE*	FALSE	--	--	BOOL
	* Save to Flash Memory Attention: Cyclic writing to FLASH Memory could damage it!					
wVal	0	65535	--	--	--	WORD
	Written value					
iErrCode			--	--	--	INT
	0: o.k. 3: invalid group or index 5: no such group 8: no such index 11: read only parameter 12: writable only at standstill 13: written value below minimum 14: written value over maximum					

Note:
ParRead and **ParWrite** provide the same scaling as the values displayed on panel or software tool. These blocks don't store their values into the FLASH while writing to the firmware parameters.

Attention:
Writing to signals (not writable!) generate *F548 FW failure*. Diagnosis 9.11 shows the relevant parameter. In this case switch off electronics and switch-on once again!

PosCorrect

Function block **PosCorrect** gets the actual position of the selected encoder. Then it is possible to set or shift the encoder position.



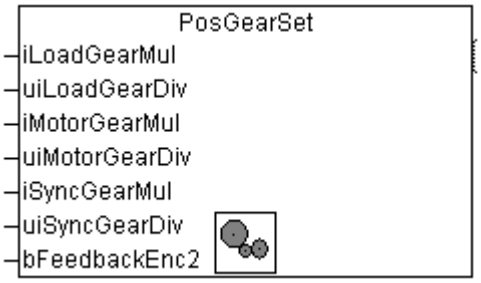
Name	LL	UL	Def	Scale	Unit	Type
bEncSel	FALSE	TRUE	FALSE	--	--	BOOL
	FALSE: Encoder 1 selected TRUE: Encoder 2 selected					
lPosValue	-2147483648	2147483647	--	--	--	DINT
	Position value					
bSetPos	FALSE	TRUE	FALSE	--	--	BOOL
	Rising edge: The actual position value of the selected encoder is set to *lPosValue*					
bShiftPos	FALSE	TRUE	FALSE	--	--	BOOL
	Rising edge: The actual position value of the selected encoder is shifted by *lPosValue*					
lPositionCorr1	-2147483648	2147483647	--	--	--	DINT
	Additional position variable which is to be set / shifted consistent with the actual position					
lPositionCorr2	-2147483648	2147483647	--	--	--	DINT
	Additional position variable which is to be set / shifted consistent with the actual position					
lPositionCorr3	-2147483648	2147483647	--	--	--	DINT
	Additional position variable which is to be set / shifted consistent with the actual position					
lActPos	-2147483648	2147483647	--	--	--	DINT
	Actual encoder position					
bPosCorrected	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Correction completed. Intended to reset the source of *bShiftPos* rep. *bSetPos*					

Note:
The format of the encoder values is according to the encoder position counter mode (P50.07 Position Counter Mode).

0	Edges	1 == 1 pulse edge
1	Scaled	65536 == 1 revolution
2	Rollover	65536 == 1 revolution, rollover axis

PosGearSet

Function block **PosGearSet** is used to implement a gear ratio in the position counter calculation.



Name	LL	UL	Def	Scale	Unit	Type
iLoadGearMul	-32768	32767	1	--	--	INT
	Encoder 1 position load gear nominator: load revolutions					
uiLoadGearDiv	1	32767	1	--	--	UINT
	Encoder 1 position load gear denominator: motor revolutions					
iMotorGearMul	-32768	32767	1	--	--	INT
	Encoder 1 motor gear nominator: motor revolutions					
uiMotorGearDiv	1	32767	1	--	--	UINT
	Encoder 1 motor gear denominator: encoder 1 revolutions					
iSyncGearMul	-32768	32767	1	--	--	INT
	Encoder 2 sync gear nominator: synchron revolutions					
uiSyncGearDiv	1	32767	1	--	--	UINT
	Encoder 2 sync gear denominator: encoder 2 revolutions					
bFeedbackEnc2	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Function block **Position** uses encoder 2 as feedback					

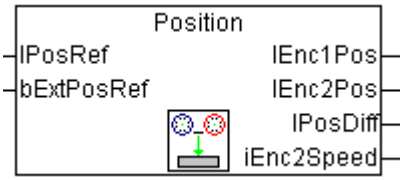
Note:

1. Load-Gear Encoder 1 position load gear: $\left(\text{gear ratio} = \frac{\text{motor shaft}}{\text{load}} \right)$
2. Motor-Gear Encoder 1 motor gear: $\left(\text{gear ratio} = \frac{\text{encoder 1}}{\text{motor shaft}} \right)$
3. Sync-Gear Encoder 2 sync gear: $\left(\text{gear ratio} = \frac{\text{encoder 2}}{\text{encoder 1}} \right)$

Only one instance of this function block may exist in an application!

Position

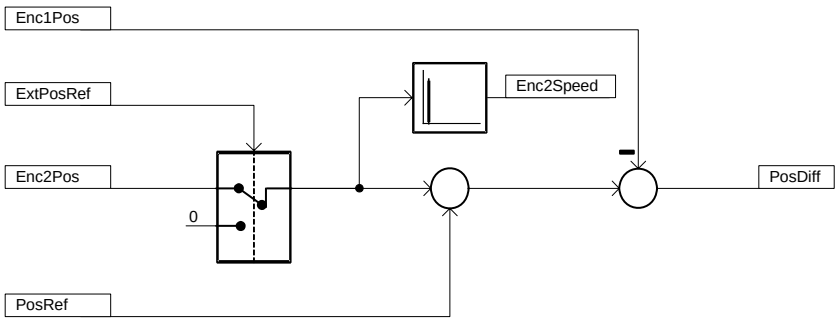
Function block **Position** is used to read the actual encoder value of encoder 1 and encoder 2 (RTAC option module). This function block can be used for position control.

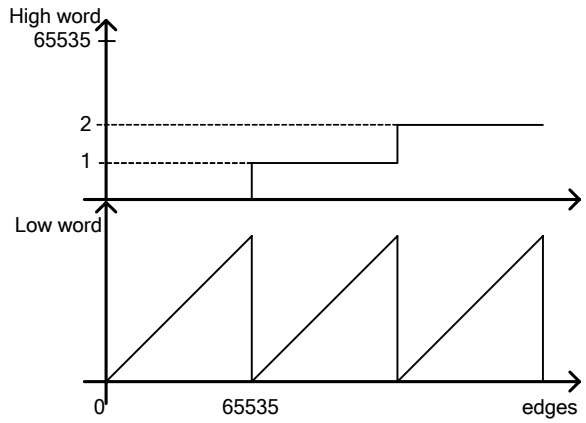
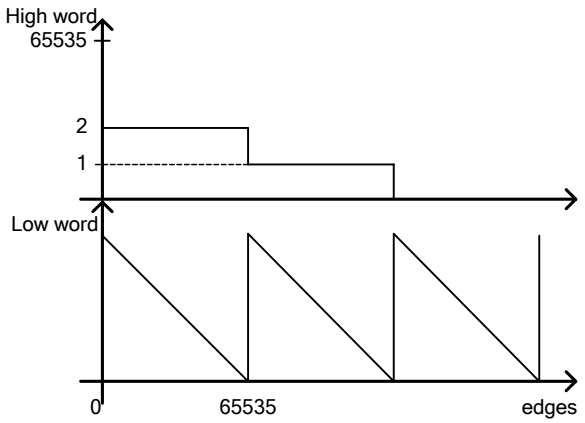
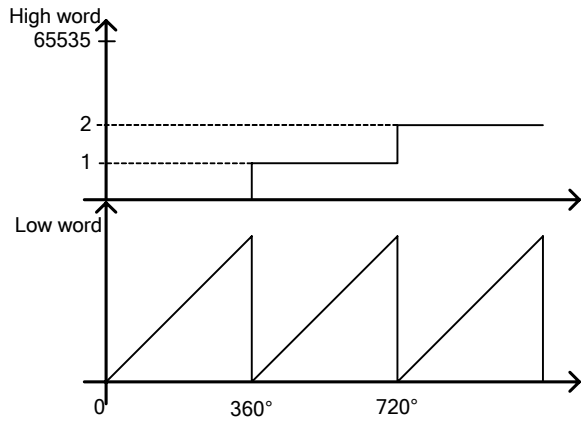
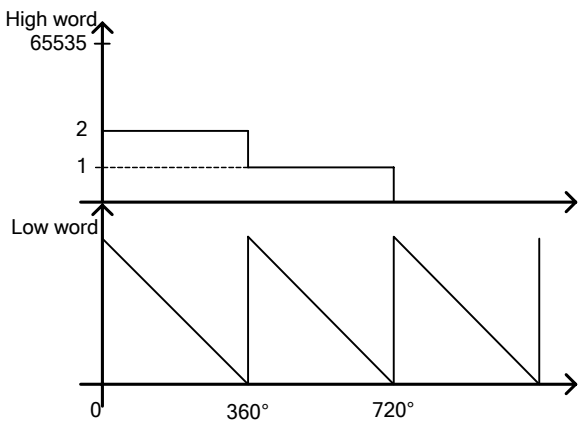
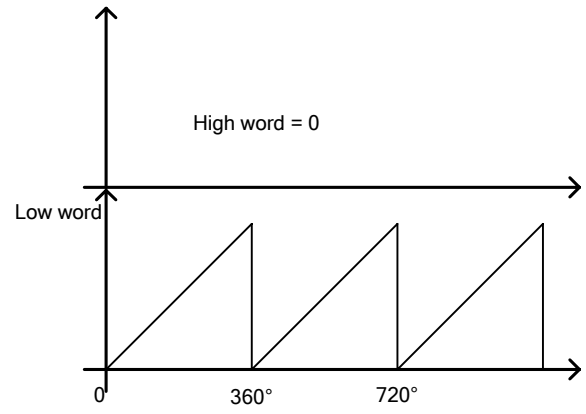
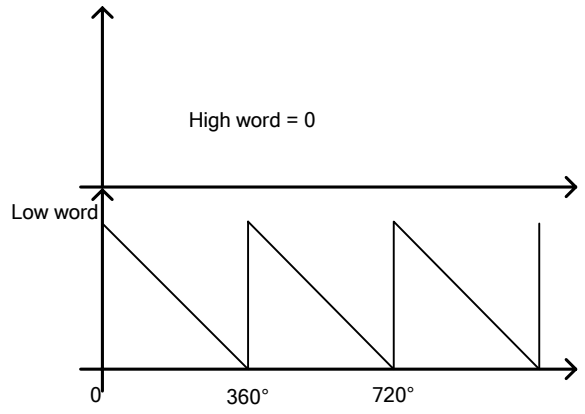


Name	LL	UL	Def	Scale	Unit	Type
lPosRef	-2147483648	2147483647	0	--	--	DINT
	Position reference (selected with input *bExtPosRef*)					
bExtPosRef	FALSE	TRUE	FALSE	--	--	BOOL
	External position reference FALSE: Read Encoder 2 Position (directly from parameter P3.05 and P3.06) TRUE: Only input *lPosRef* selected					
lEnc1Pos	-2147483648	2147483647	0	--	--	DINT
	Encoder 1 position					
lEnc2Pos (3.05, 3.06)	-2147483648	2147483647	0	--	--	DINT
	Encoder 2 position If *bExtPosRef* = TRUE: *iEnc2Pos* = 0					
lPosDiff	-2147483648	2147483647	0	--	--	DINT
	Position difference *bExtPosRef* = FALSE: *lPosDiff* = *lPosRef* + *lEnc2Pos* – *lEnc1Pos* *bExtPosRef* = TRUE: *lPosDiff* = *lPosRef* – *iEnc1Pos*					
iEnc2Speed	-32768	32767	0	--	--	INT
	Calculated speed from encoder 2 (scaled in standard speed scaling) If *bExtPosRef* = TRUE: *iEnc2Speed* = 0					

Note:
Scaling of position outputs according to parameter P50.07 of DCS800 firmware!

Circuit diagram:

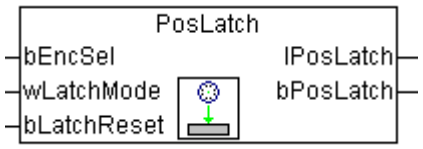


Turn right (proposed forward direction, using bridge 1)	**Turn left** (reverse direction, using bridge 2)
P50.07: Pulse Edges 	
P50.07: Scaled 	
P50.07: Rollover 	

Note:
For further information see DCS800 firmware manual group 50!

PosLatch

Function block **PosLatch** can be used for position latch by using a touch probe.



Name	LL	UL	Def	Scale	Unit	Type
bEncSel	FALSE	TRUE	FALSE	--	--	BOOL
	FALSE: Encoder 1 selected TRUE: Encoder 2 selected					
wLatchMode			--	--	--	WORD
	Encoder position latch enable: 0: Not used 1: Rising edge of DI1 (CON-4) 2: Falling edge of DI1 (CON-4) 3: Zero pulse of encoder 1 4: Zero pulse of encoder 1 and DI1 = High (CON-4) 5: Zero pulse of encoder 1 and DI1 = Low (CON-4) Encoder 2 position latch enable: 0: Not used 1: Rising edge of X2:1-2 (on RTAC-03 module only, not available on RTAC-01 module) 2: Falling edge of X2:1-2 (on RTAC-03 module only, not available on RTAC-01 module) 3: zero pulse of encoder 2 4: Zero pulse of encoder 2 && X2:1-2 = High (on RTAC-03 module only, not available on RTAC-01 module) 5: Zero pulse of encoder 2 && X2:1-2 = Low (on RTAC-03 module only, not available on RTAC-01 module)					
bLatchReset	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: reset output signal *bPosLatch*					
lPosLatch	-2147483648	2147483647	--	--	--	DINT
	Latched encoder position					
bPosLatch	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Encoder position has been latched					

Note:
The format of the encoder values is according to the encoder position counter mode (P50.07 Position Counter Mode).

0	Edges	1 == 1 pulse edge
1	Scaled	65536 == 1 revolution
2	Rollover	65536 == 1 revolution, rollover axis

This function should be executed in 5 ms task level!

ProgProtect

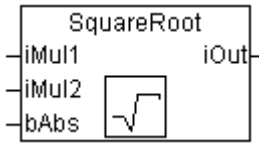
Function block **ProgProtect** can be used to protect an application program code, stored on the Memory Card, against copying.

Name	LL	UL	Def	Scale	Unit	Type
bProtect	FALSE	TRUE	FALSE	--	--	BOOL
	FALSE: no action TRUE: Uploading of program and source code not possible					

Note:
Add this block to your initialisation task.

SquareRoot

Function block **SquareRoot** includes a time optimised function to calculate square root.



$$iOut = \sqrt{iMul1 \cdot iMul2}$$

Name	LL	UL	Def	Scale	Unit	Type
iMul1	-32768 Input 1	32767	--	--	--	INT
iMul2	-32768 Input 2	32767	--	--	--	INT
bAbs	FALSE	TRUE*	FALSE	--	--	BOOL
	* Negative values on *iMul1* and *iMul2* will be accepted as absolute number					
iOut	0 Result of square root calculation	32767	--	--	--	INT

SquareRoot2

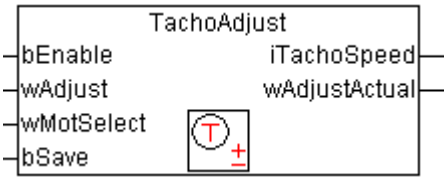
Function block **SquareRoot2** includes a time optimized function to calculate square root.

$$iOut = \sqrt{lInput}$$

Same functionality like function block **SquareRoot**. Difference is only the input as 32 bit value!

TachoAdjust

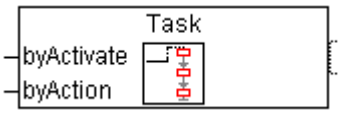
Function block **TachoAdjust** can be used to do a tacho adjustment via application program. This function is similar to tacho fine tuning in DCS800 Start-up assistant.



Description on request

Task

Function block **Task** can be used to activate a single shot for a task. This task isn't executed cyclic!

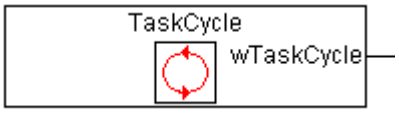


Name	LL	UL	Def	Scale	Unit	Type
byActivate	0 0: no action 1: task action according to byAction	1	0	--	--	BYTE
byAction	1: start user action 1 2: start user action 2 3: start user action 3 4: start user action 4 5: start freewheeling task 6: stop freewheeling task		--	--	--	BYTE

Note:
These tasks are definable in the CoDeSys task configuration.

TaskCycle

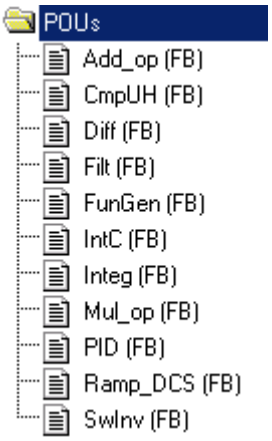
Function block **TaskCycle** is used to read the cycle time of the current task in which the function block is working.
This is needed for the definition of new function blocks if a time relationship should exist.



Name	LL	UL	Def	Scale	Unit	Type
wTaskCycle	0 Actual cycle time	65535	--	--	ms	WORD

Arithmetic

Name: **DCS800arithmetic.lib**
Version: **1.20**
Firmware: **Don't care!**



Add_op

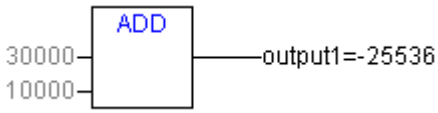
The function **Add_op** provides a limitation to avoid wrap around. It works in the range –32767…+32767 and limits the result.

$$iIn1 + iIn2 = iADD\_OP$$

Name	LL	UL	Def	Scale	Unit	Type
iIn1	-32767 Input 1	32767	--	--	--	INT
iIn2	-32767 Input 2	32767	--	--	--	INT
iADD_OP	-32767 Output	32767	--	--	--	INT

Example:

Function block **ADD** (Operator):



→ typical overflow

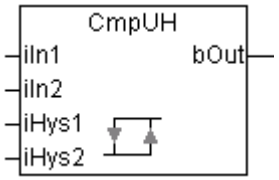
Function block **Add_op** (Arithmetic Library):

→ output limitation (- 32767…+ 32767)

Note:
With IEC 61131 the function block **ADD** generates an overflow which creates negative values for results above 32767.
In this case of position counter use the function block **Mul_op** or **MulDiv**.

CmpUH

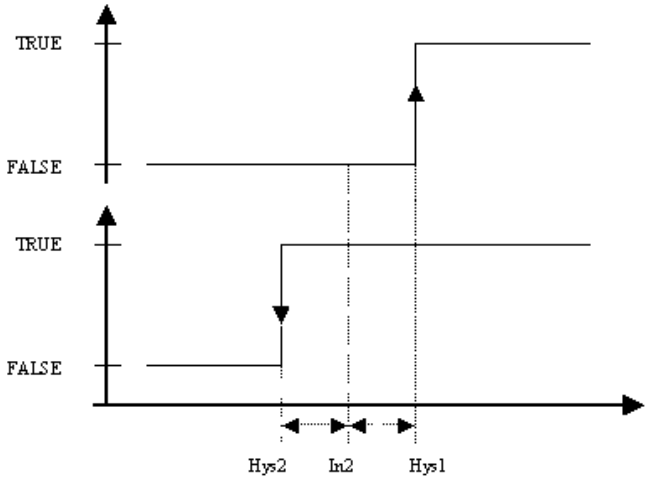
Function block **CmpUH** is a comparator with two asymmetrical hystereses.



$$bOut = TRUE : (iIn - iIn2 - iHys1) \geq 0$$
$$bOut = FALSE : (in1 - iIn2 + iHys2) < 0$$

Name	LL	UL	Def	Scale	Unit	Type
iIn1	0 Input 1	32767	--	--	--	INT
iIn2	0 Input 2	32767	--	--	--	INT
iHys1	0 Hysteresis 1	32767	0	--	--	INT
iHys2	0 Hysteresis 2	32767	0	--	--	INT
bOut	FALSE Output	TRUE	FALSE	--	--	BOOL

Switch points and hysteresis of function block **CmpUH**:



Note:
For comparator functions also use the function blocks:

- **GT** / **GE** (Greater than / greater equal)
- **LT** / **LE** (Lower than / lower equal)
- **EQ** (Equal)

These blocks are CoDeSys operators!

Diff

Function block **Diff** is a discrete differentiator for integer values.

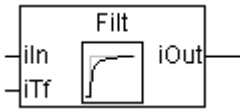
Name	LL	UL	Def	Scale	Unit	Type
iIn	-32767 Input	32767	--	--	--	INT
iTd	0 Differentiation time	30000	1000	1 = 1 ms	ms	INT
iOut	-32767 Output	32767	--	--	--	INT

Step response with function block **Diff**:

Note:
The output value is limited by integer range (-32767…32767).

Filt

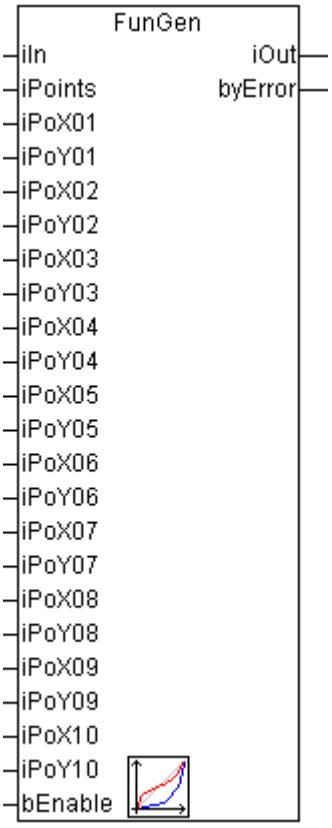
Function block **Filt** is a discrete low-pass filter (PT-1 characteristic).



Name	LL	UL	Def	Scale	Unit	Type
iIn	-32768 Input	32767	--	--	--	INT
iTf	0 > 0: filter time = 0: bypass (*iIn = iOut*)	30000	1000	1 = 1 ms	ms	INT
iOut	-32767 Output	32767	--	--	--	INT

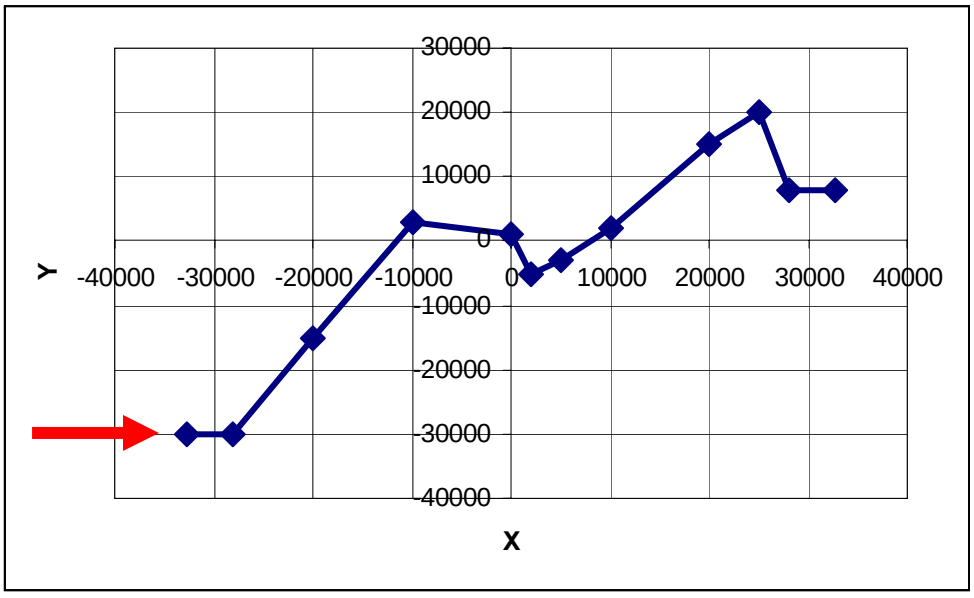
FunGen

The function block **FunGen** is a function generator with an X-Y-coordinate system. There are 10 supporting points maximum to create a curve. All 4 quadrants can be used!



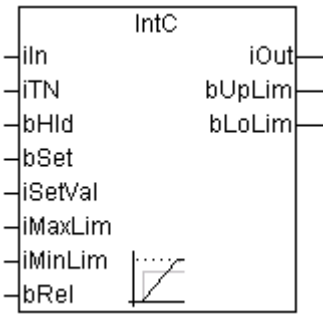
Name	LL	UL	Def	Scale	Unit	Type
iIn	-32768	32767	--	--	--	INT
	Input for the x-signal					
iPoints	2	10	2	--	--	INT
	Available points, beginning with point 1					
iPoX01	-32768	32767	--	--	--	INT
	1. x-point					
iPoY01	-32768	32767	--	--	--	INT
	1. y-point					
iPoX02	-32768	32767	--	--	--	INT
	2. x-point					
iPoY02	-32768	32767	--	--	--	INT
	2. y-point					
iPoX03	-32768	32767	--	--	--	INT
	3. x-point					
iPoY03	-32768	32767	--	--	--	INT
…	3. y-point					
bRel	FALSE	TRUE *	FALSE	--	--	BOOL
	* Enable function block; FALSE: *iOut* = 0					
iOut	-32767	32767	--	--	--	INT
	Output for the Y-signal					
byError	0	3	0	--	--	BYTE
	0: *iOut* is within working range 1: minimum value is reached 2: maximum value is reached 3: *iPoints* out of range					

Note:
Point iPoX01 and iPoY01 must be the first supporting point. The last supporting point will be set with input value iPoints (2…10).



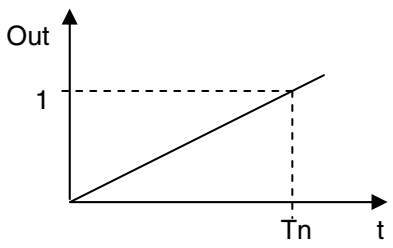
IntC

The function block **IntC** is an integrator with the features: set, hold, reset and a selectable limit



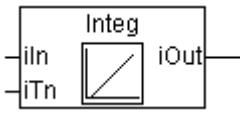
Name	LL	UL	Def	Scale	Unit	Type
iIn	-32768	32767	--	--	--	INT
	Input for reference					
iTN	0	30000	1000	1 = 1 ms	ms	INT
	> 0:Integration time constant = 0 : Output is set to zero					
bHld	FALSE	TRUE *	FALSE	--	--	BOOL
	Hold on the integrator					
bSet	FALSE	TRUE *	FALSE	--	--	BOOL
	Input to set a SetVal to the output					
iSetVal	-32768	32767	0	--	--	INT
	Defined value					
iMaxLim	0	30000	30000	--	--	INT
	Value for maximum level					
iMinLim	-30000	0	-30000	--	--	INT
	Value for minimum level					
bRel	FALSE	TRUE *	FALSE	--	--	BOOL
	Enable or disable the output (reset)					
iOut	-32767	32767	--	--	--	INT
	Output of the integrator					
bUpLim	FALSE	TRUE *	FALSE	--	--	BOOL
	Output is at upper level					
bLoLim	FALSE	TRUE *	FALSE	--	--	BOOL
	Output is at lower level					

Step response of function block **IntC**:



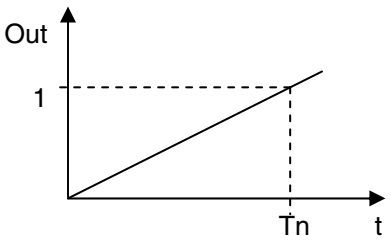
Integ

Discrete Integrator for Integer values.



Name	LL	UL	Def	Scale	Unit	Type
iIn	-32768 Input	32767	--	--	--	INT
iTn	0 > 0: Integration time = 0: Output is set to zero	30000	1000	1 = 1 ms	ms	INT
iOut	-32767 Output	32767	--	--	--	INT

Step response with function block **Integ**:



Note:
The output is limited to –32767…+32767.

Mul_op

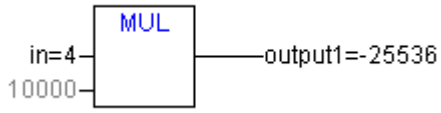
The function **Mul_op** provides a limitation to avoid wrap around. It works in the range +32767 until –32767 and limits the result.

$$iIn1 \cdot iIn2 = iMUL\_OP$$

Name	LL	UL	Def	Scale	Unit	Type
iIn1	-32768 Input 1	32767	0	--	--	INT
iIn2	-32768 Input 2	32767	0	--	--	INT
iMul_op	-32767 Output	32767	0	--	--	INT

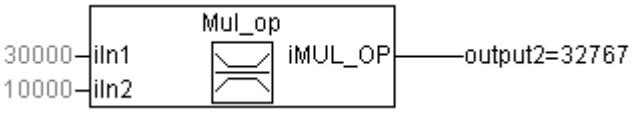
Example:

Function block **MUL** (Operator):



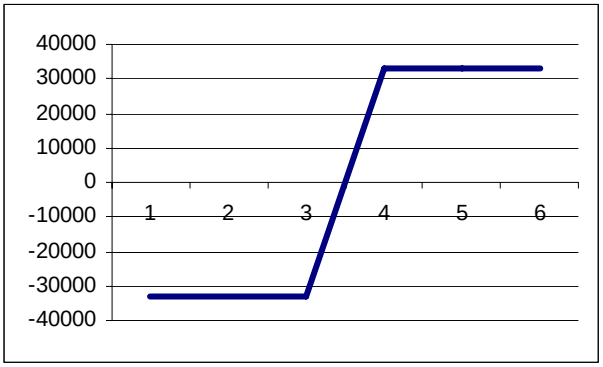
→ typical overflow

Function block **Mul_op** (Arithmetic Library):



→ output limitation (- 32767…+ 32767)

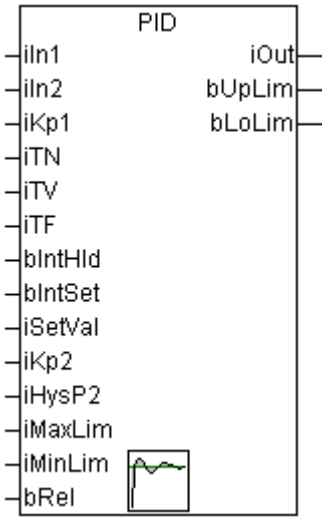
Operating range:



Note:
With IEC 61131 the function block **MUL** generates an overflow which creates negative values for results above 32767.
In this case of position counter use the function block **Mul_op** or **MulDiv**.

PID

The function block **PID** is a discrete PID-Controller.
This block can also be used as a P-, PI- or PD-Controller.



Controller error (normal):

$$error = iIn1 - iIn2$$

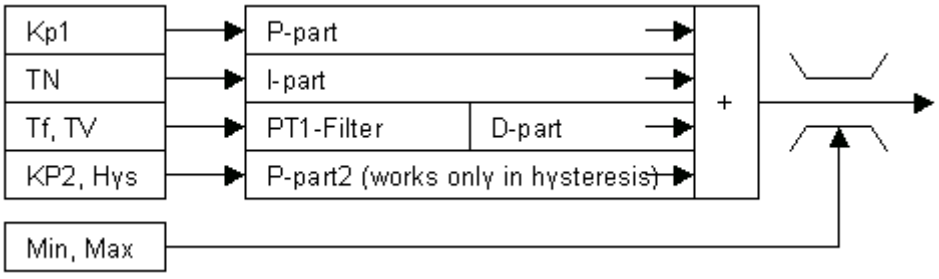
Controller error (for second P-part):

$$Kp2\,error = iIn1 - iIn2 - iHysP2$$

The PID-Controller is based on following algorithm:

$$G(s) = \left[Kp1 + \frac{Kp1}{TNs} + Kp1 \cdot \frac{TVs}{1 + TFs} + Kp2 \right]$$

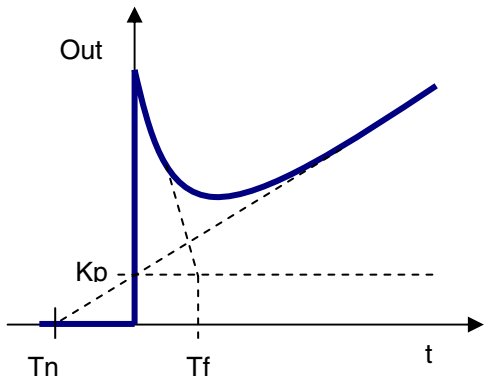
Note:
The PID-controller works like the following picture:



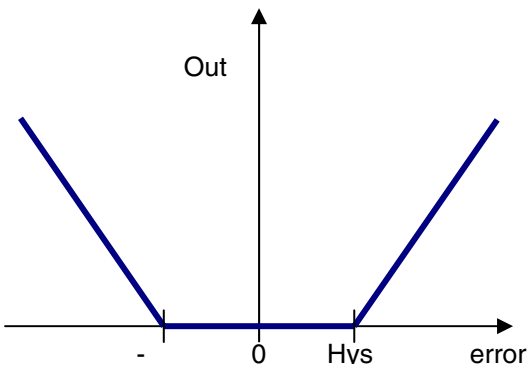
The output is limited by *iMaxLim* and *iMinLim*.
Enable controller with input *bRel*.

Name	LL	UL	Def	Scale	Unit	Type
iIn1	-32768	32767	--	--	--	INT
	Input for reference value					
iIn2	-32768	32767	--	--	--	INT
	Input for actual value					
iKp1	0	32767	500	100 = 1	--	INT
	P-part 1 (gain)					
iTN	0	32767	1000	1 = 1 ms	ms	INT
	I-part (disable with 0)					
iTV	0	32767	0	1 = 1 ms	ms	INT
	D-part (disable with 0)					
iTF	1	30000	0	1 = 1 ms	ms	INT
	Filter time for D-part					
iKp2	0	32767	5	100 = 1	--	INT
	P-part 2, working only outside hysteresis range set with *iHysP2*, disable with 0					
iHysP2	0	20000	0	--	--	INT
	Hysteresis for P-part 2					
iSetVal	-32768	32767	0	--	--	INT
	Set value for I-part memory					
iMaxLim	0	30000	30000	--	--	INT
	Maximum limitation					
iMinLim	-30000	0	-30000	--	--	INT
	Minimum limitation					
bIntHld	FALSE	TRUE *	FALSE	--	--	BOOL
	* Hold I-part memory					
bIntSet	FALSE	TRUE *	FALSE	--	--	BOOL
	* Set I-part memory to *iSetVal*					
bRel	FALSE	TRUE *	TRUE	--	--	BOOL
	* Enable controller, FALSE: I-part memory = 0, *iOut* = 0					
iOut	-32767	32767	--	--	--	INT
	Controller output					
bUpLim	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Maximum limitation is reached					
bLoLim	FALSE	TRUE	FALSE	--	--	BOOL
	TRUE: Minimum limitation is reached					

Step response of function block **PID**:

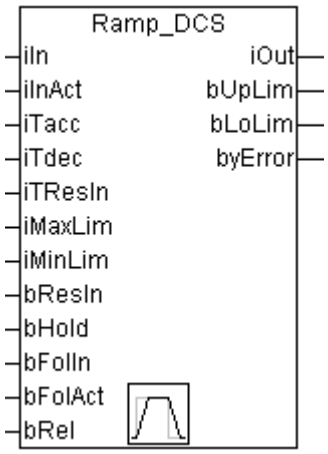


Working range of second P-part:



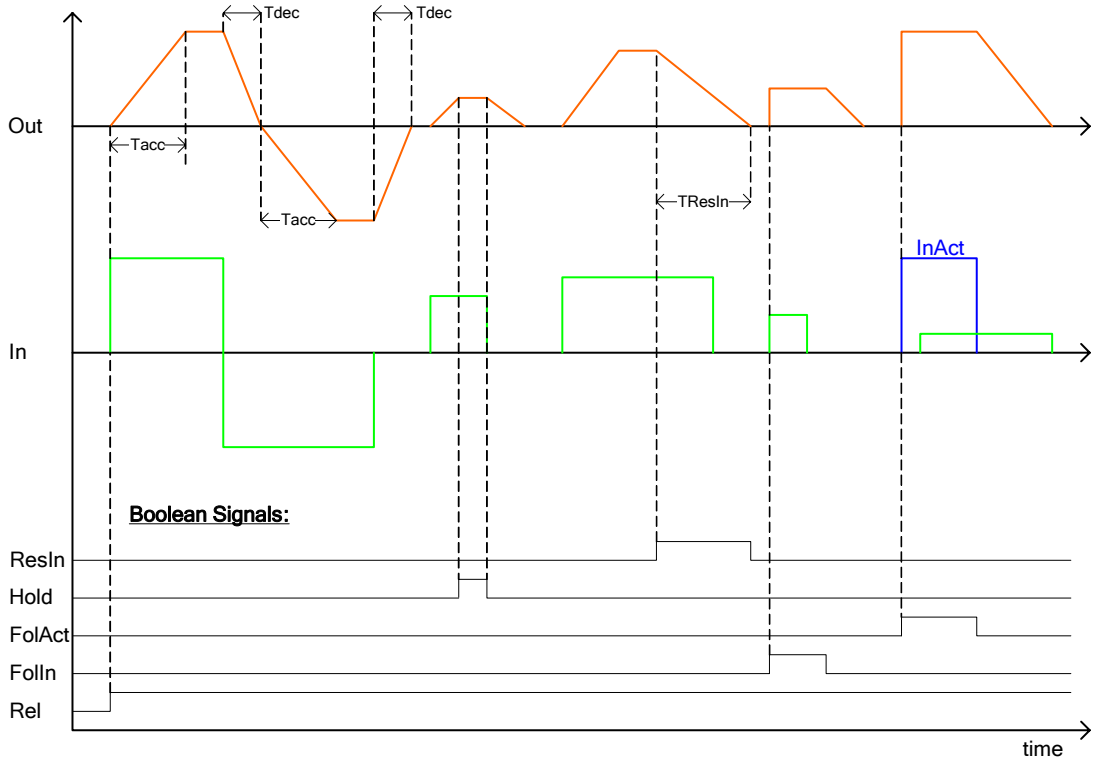
Ramp_DCS

The function block **Ramp_DCS** is a ramp generator with the features: ramp time up, ramp time down, hold, set back or set to a defined value.



$$iOut = iOut + \frac{Task \cdot 1000}{Time}$$ (cyclic executed!)

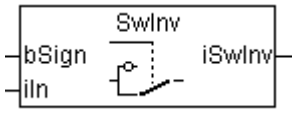
Note:
The diagram shows the functionality of the ramp generator.



Name	LL	UL	Def	Scale	Unit	Type
iIn	-32767	32767	--	--	--	INT
	Reference input					
iInAct	-32767	32767	0	--	--	INT
	Input for the value that is set with command *bFolAct*					
iTacc	0	30000	1000	1=10ms	ms	INT
	Upward ramp time					
iTdec	0	30000	1000	1=10ms	ms	INT
	Downward ramp time					
iTResIn	0	30000	1000	1=10ms	ms	INT
	Downward ramp time with active *bResIn*					
iMaxLim	iMinLim	32767	32767	--	--	INT
	Input for the maximum value					
iMinLim	-32767	iMaxLim	-32767	--	--	INT
	Input for the minimum value					
bResIn	FALSE	TRUE *	FALSE	--	--	BOOL
	Input to enable the special downward ramp to zero with time *iTResIn*					
bHold	FALSE	TRUE *	FALSE	--	--	BOOL
	Command to stop the ramp generator and hold the value					
bFolIn	FALSE	TRUE *	FALSE	--	--	BOOL
	Input to set the input *iIn* directly to the output (bypass)					
bFolAct	FALSE	TRUE *	FALSE	--	--	BOOL
	Input to set the value *iInAct* to the output					
bRel	FALSE	TRUE *	TRUE	--	--	BOOL
	* Enable function block, FALSE: *iOut* = 0					
iOut	-32767	32767	--	--	--	INT
	Output value of function block					
bUpLim	FALSE	TRUE	FALSE	--	--	BOOL
	Signal has been reached maximum level					
bLoLim	FALSE	TRUE	FALSE	--	--	BOOL
	Signal has been reached minimum level					

SwInv

With this function block the sign of a value can be changed.

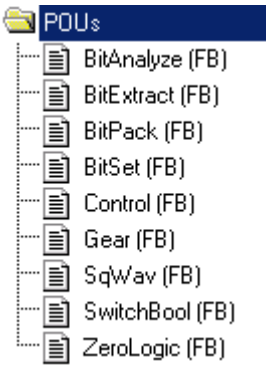


$$bSign = \mathrm{TRUE}: \; -\; iIn = SwInv$$
$$bSign = \mathrm{FALSE}: \; iIn = SwInv$$

Name	LL	UL	Def	Scale	Unit	Type
bSign	FALSE	TRUE	FALSE	--	--	BOOL
	Switch input					
iIn	-32767	32767	--	--	--	INT
	Input					
iSwInv	-32767	32767	--	--	--	INT
	Output					

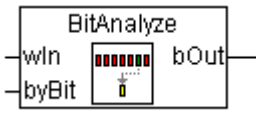
Special

Name: **DCS800special.lib**
Version: **1.10**
Firmware: **Don't care!**



BitAnalyze

The function block **BitAnalyze** is used to analyze a selected bit from a data word. At the output the status of the selected bit is shown.

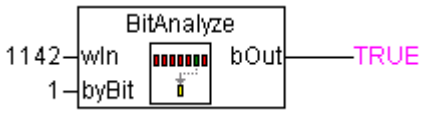


Name	LL	UL	Def	Scale	Unit	Type
wIn	0 Input value	65535	--	--	--	WORD
byBit	0 Selected bit	255	--	--	--	BYTE
bOut	FALSE Boolean output	TRUE	FALSE	--	--	BOOL

Example:

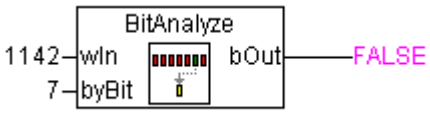
$1142_{dec} \Leftrightarrow 476_{hex} \Leftrightarrow 0100\,|\,0111\,|\,0110_{bin}$

Bit 1 should be analyzed:



→ Bit 1 is *TRUE*

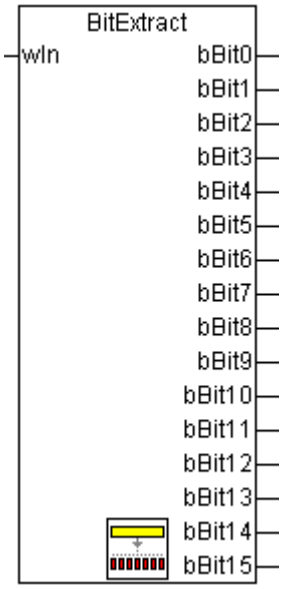
Bit 7 should be analyzed:



→ Bit 7 is *FALSE*

BitExtract

With function block **BitExtract** there is the possibility to unpack a word value into the several bits.

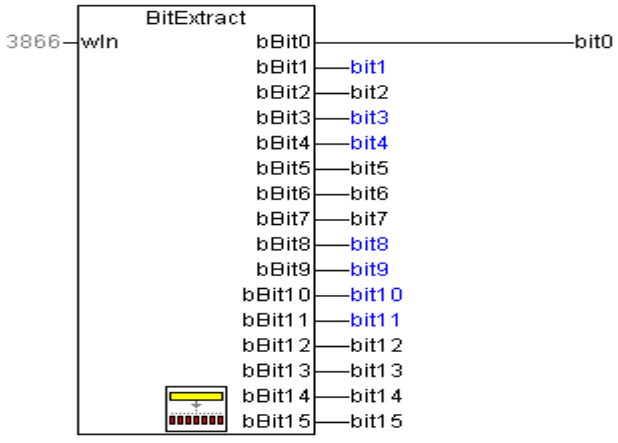


Name	LL	UL	Def	Scale	Unit	Type
wIn	0	65535	--	--	--	WORD
	Input value in word format					
bBit…	FALSE	TRUE	FALSE	--	--	BOOL
	Actual value of bit …					

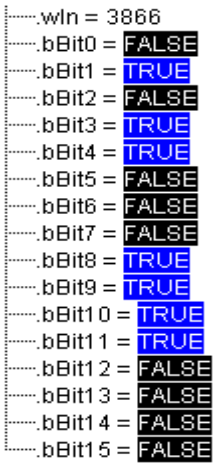
Example:

$3866_{dec} \Leftrightarrow F1A_{hex} \Leftrightarrow 1111|0001|1010_{bin}$

Function block **BitExtract**:

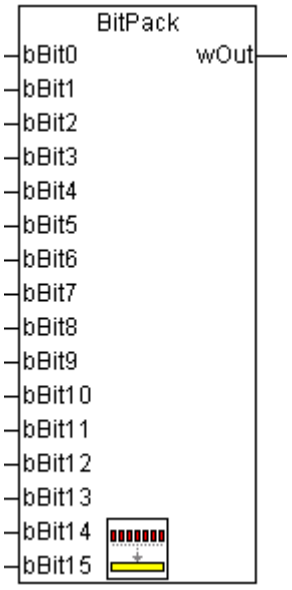


Declaration line:



BitPack

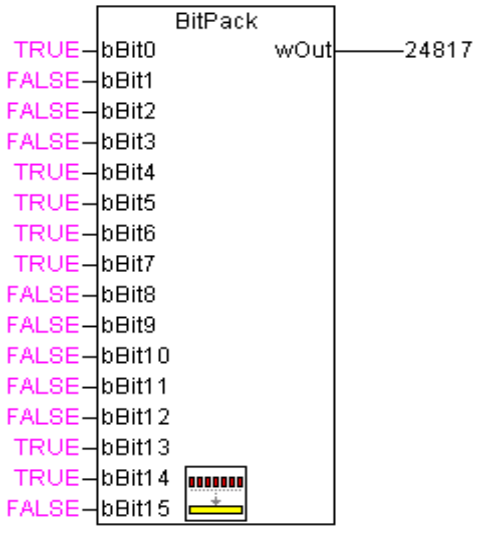
With function block **BitPack** it is possible to pack several bits in word format.



Name	LL	UL	Def	Scale	Unit	Type
bBit…	FALSE	TRUE	FALSE	--	--	BOOL
…	Input value of bit…					
wOut	0	65535	--	--	--	WORD
	Output value packed					

Example:

$$24817_{dec} \Leftrightarrow 60F1_{hex} \Leftrightarrow 0110\,|\,0000\,|\,1111\,|\,0001_{bin}$$



BitSet

The function block **BitSet** is used to set a bit in a word.

Name	LL	UL	Def	Scale	Unit	Type
wIn	0 Input value	65535	--	--	--	WORD
byBit	0 Selected bit, beginning with 0	15	--	--	--	BYTE
bVal	FALSE Value for selected bit	TRUE	FALSE	--	--	BOOL
wOut	0 Output value	65535	--	--	--	WORD

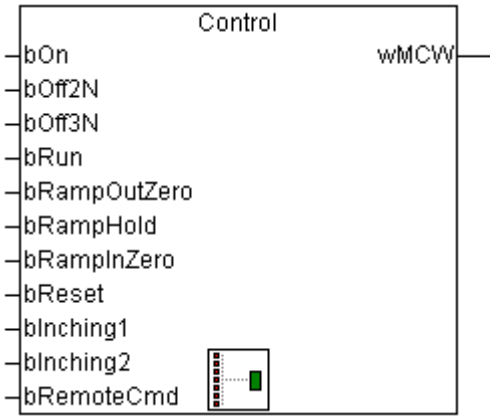
Example:

Bit 5 of input should be set. It is normally *FALSE*!

The output shows the input value with the manipulate bit number 5!

Control

This is a function block to create the *Main Control Word* (MCW) and match the format of *MCW* (P 7.01).

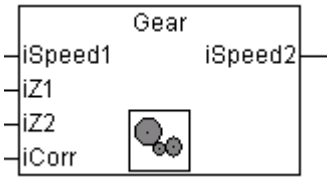


Name	LL	UL	Def	Scale	Unit	Type
bOn	FALSE	TRUE *	FALSE	--	--	BOOL
	* Command to RdyRun state (switch main contactor on)					
bOff2N	FALSE	TRUE *	FALSE	--	--	BOOL
	* No Off2 (coast stop)					
bOff3N	FALSE	TRUE *	FALSE	--	--	BOOL
	* No Off3 (E-Stop)					
bRun	FALSE	TRUE *	FALSE	--	--	BOOL
	* Command to RdyRef state					
bRampOutZ-ero	FALSE *	TRUE	FALSE	--	--	BOOL
	* Speed ramp output is forced to zero					
bRampHold	FALSE *	TRUE	FALSE	--	--	BOOL
	* Freeze (hold) speed ramp					
bRampInZero	FALSE *	TRUE	FALSE	--	--	BOOL
	* Speed ramp input is forced to zero					
bReset	FALSE	TRUE *	FALSE	--	--	BOOL
	* Acknowledge fault indications with the positive edge					
bInching1	FALSE	TRUE *	FALSE	--	--	BOOL
	* Constant speed defined by FixedSpeed1 (23.02)					
bInching2	FALSE	TRUE *	FALSE	--	--	BOOL
	* Constant speed defined by FixedSpeed2 (23.03)					
bRemoteCmd	FALSE	TRUE *	FALSE	--	--	BOOL
	* Overriding control enabled					
wMCW	0	65535	--	--	--	WORD
	Output *Main Control Word* packed					

Note:
Unused inputs are set to FALSE

Gear

The **Gear** function calculates a gear box transmission. Additional feature is the correction input.



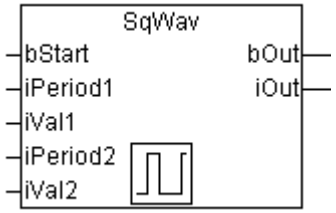
$$iSpeed2 = \frac{iSpeed1 \cdot iZ1 + iCorr}{iZ2}$$

Name	LL	UL	Def	Scale	Unit	Type
iSpeed1	-32768 Input speed	32767	--	--	--	INT
iZ1	-32768 Gearwheel 1	32767	1	--	--	INT
iZ2	-32768 Gearwheel 2	32767	1	--	--	INT
iCorr	-32768 Correction factor	32767	0	--	--	INT
iSpeed2	-32767 Output speed	32767	--	--	--	INT

Note:
The output is limited to – 32767…+ 32767.

SqWav

The function block **SqWav** generates a square wave on the output. Time and magnitude are parameters.

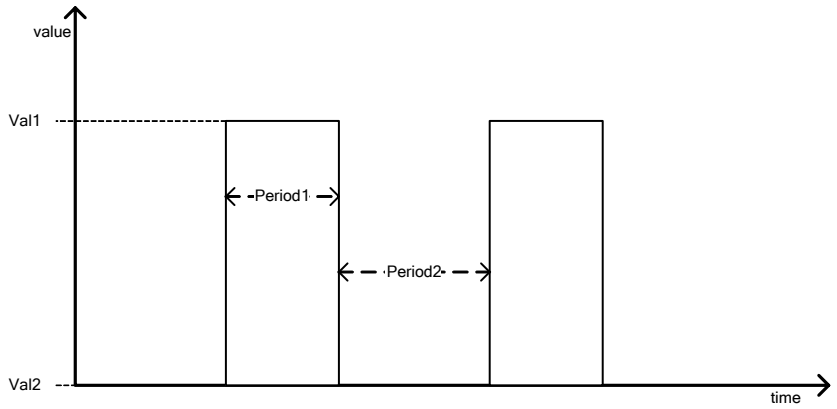


If *bStart* = TRUE → function block will be enabled

Name	LL	UL	Def	Scale	Unit	Type
bStart	FALSE * Enable function block	TRUE*	FALSE	--	--	BOOL
iPeriod1	0 Time of period 1	32767	1000	1 = 1 ms	ms	INT
iVal1	-32768 Value of period 1	32767	20000	--	--	INT
iPeriod2	0 Time of period 2	32767	1000	1 = 1 ms	ms	INT
iVal2	-32768 Value of period 2	32767	-20000	--	--	INT
bOut	FALSE Output as Boolean value	TRUE	FALSE	--	--	BOOL
iOut	-32768 Output as Integer value	32767	--	--	--	INT

Example:

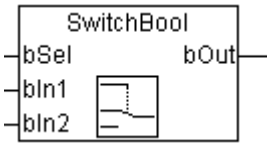
The picture shows the diagram for an integer signal which alternates between *iVal1* and *iVal2*. Input *iPeriod1* and *iPeriod2* defines the time for the waves.



Note:
If *Start* = TRUE the wave starts with *Period1* and *Val1*.

SwitchBool

With function block **SwitchBool** there is the possibility to switch between Boolean values.



If *bSel* = TRUE → *bOut* = *bIn2*
If *bSel* = FALSE → *bOut* = *bIn1*

Name	LL	UL	Def	Scale	Unit	Type
bSel	FALSE	TRUE	FALSE	--	--	BOOL
	Selector input for *bIn1* or *bIn2*					
bIn1	FALSE	TRUE	FALSE	--	--	BOOL
	Value of input 1					
bIn2	FALSE	TRUE	FALSE	--	--	BOOL
	Value of input 2					
bOut	FALSE	TRUE	FALSE	--	--	BOOL
	Output value					

Index of function blocks

Appendix B – Fault Tracing

Overview

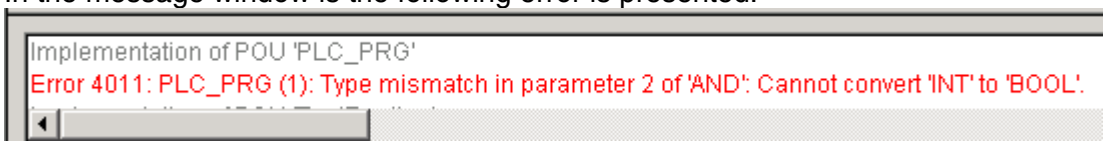
There are two types of errors. The first category are errors which stick together with the CoDeSys software tool. Other errors are concerned with the DCS800 drive.

CoDeSys software errors

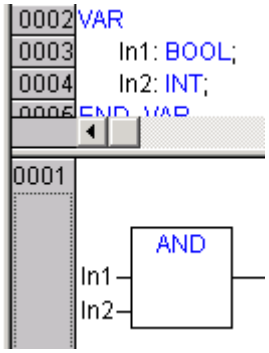
Errors in a program will be shown in the message window on the desktop. Errors are shown in red and provide an error number. These numbers are documented in the *CoDeSys User Manual V2.3* which has been installed automatically on your PC. Only if there are no errors in the program is it possible to download a program into the drive.

Example:

In the message window is the following error is presented:



Click on the error and the screen jumps to the faulty line or network.



You see that input *In2* is an INT (integer) type but the function block needs a Boolean value.
→ Change the type of *In2* to solve the error.
→ Compile again!

Note:
Look up the error number in *CoDeSys User Manual V2.3* and follow the instructions. These errors are not illustrated in this ABB manual!

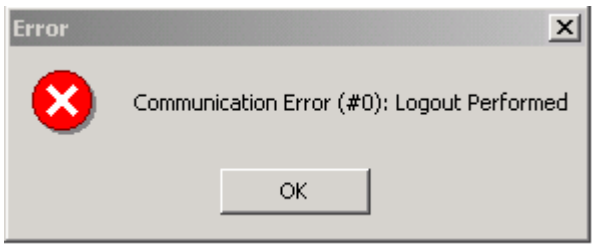
DCS800 errors

DCS800 errors will be shown in a pop-up window before download. These errors check the parameter list and the event settings. The several errors are listed below.

Error

Communication Error (#0)

Communication error means that the serial communication is faulty.



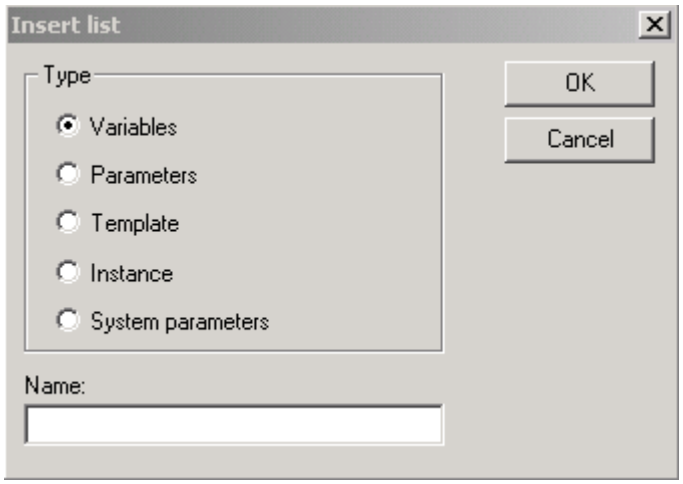
- The communication channel is incorrect
- Drive Window Light is active
- The ABB Memory Card (SDCS-MEM-8) is not in slot 4
- The DCS800 electronics is not switched-on

Error 4100

Invalid parameter list

The used parameter list includes errors or a faulty type is selected.

Note:
Only type *Variables* is possible!



Error 4101 **No memory for application parameter text**

The memory for parameter text is full. Delete other texts or define a shorter one.

Error 4102 **Too many enumerations for Parameter …**

There are only 12 characters available for a message text.

Error 4103 **Unsupported type of parameter list**

The following types are available:
- INT (integer)
- BOOL (boolean)
- BYTE (byte)
- ENUM (enumeration)

Error 4104 **More than 250 application parameters defined**

Only 250 parameters are available!

Error 4105 **Parameter … already exists**

The parameter is already in use. Declare another one.

Error 4106 **Unsupported variable type for parameter …**

The selected variable type is not available!

Error 4107 **Parameter … not settable**

The selected parameter is not settable.

Error 4108 **No valid application selected**

For download there is no application selected. This is a problem with the CoDeSys software.

Error 4109 **Flash memory access conflict; repeat last action**

The downloaded code could be corrupted because other actions worked during download time. Repeat last action!

Error 4110 **Missing task configuration**

There is no task defined. Application program is only possible with a triggering task! Freewheeling isn't possible!

Error 4111 **File upload not allowed this time**

Upload of application program is not allowed. Check if the application is protected with function block **ProgProtect** and disable the protection if necessary.

Error 4112 **Application protected**

The application is protected by password. Upload not possible!

Error 4113 **Retain data invalid**

The saved data on the Flash Memory is invalid. This error message starts when parameters of Flash Memory will be uploaded to RAM.

Error 4114 **Debug mode not allowed if drive is on**

Debug functions is not allowed if the drive is running. Change to Off mode please and start debugging.

Error 4115 **Parameter definitions loaded**

This is not an error!
Parameter definitions are loaded into the drive.

Error 4116 **Program saved**

This is not an error!
The program is saved on the Flash Memory correctly!

Error 4117 **Program deleted**

This is not an error!
The program is deleted from the Flash Memory!

Error 4118 **Reloading of parameter definitions not allowed**

Only one parameter definition can be downloaded. This can happen after online change of parameter list.
Delete the second definition (command <clean all>) and download it again.

Error 4119 **Compact Flash memory invalid**

The used Flash Memory Card is invalid. Switch-off the electronics and change the ABB Memory Card.

Error 4120 **Source code requires Compact Flash memory**

It is necessary to insert Flash Memory.

Error 4121 **Max. file size exceeded**

The used file size is too big. Reduce file size or optimize the program code!

Error 4122 **Program invalid**

The downloaded program is invalid. Check the program please!

Error 4123 **New program**

This is not an error!
A new program is activated.

Error 4124 **Compact Flash card missing**

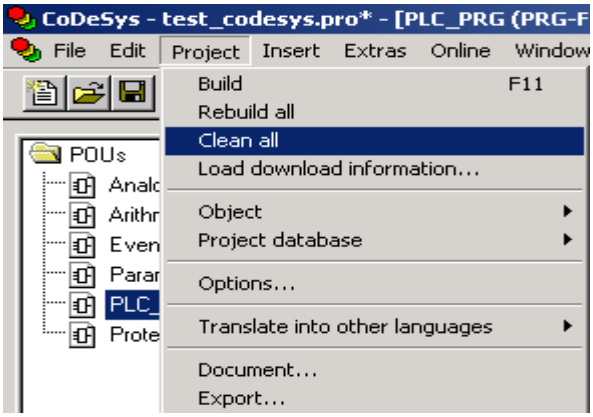
The Flash Memory card isn't in slot 4. Application program only works with a ABB MEM-8 Flash Memory card.

Error 4125 **… ms task stalled**

The task cannot be triggered, because of an endless loop.

Download error

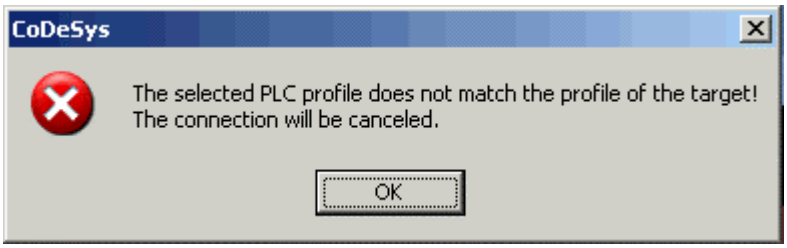
Problems during download of a program which are downloaded before can be solved with the *Clean All* function.
It removes old downloading files which are not used!



The problem can arise because of online changes in parameter manager. Use *Clean All* function.

Profile error

DCS800 needs the ABB Memory Card for communication and download. If the Memory Card isn't in the slot, download isn't possible and the connection will be cancelled!



Insert the ABB Memory Card (with the electronics switched-off) and try again!

DCS800 family

DCS800-S modules

The versatile drive for any application

20 … 5,200 A_{DC}
0 … 1,160 V_{DC}
230 … 1,000 V_{AC}
IP00

- Compact
- Highest power ability
- Simple operation
- Comfortable assistants, e.g. for commissioning or fault tracing
- Scalable to all applications
- Free programmable by means of integrated IEC61131-PLC

DCS800-A enclosed converters

Complete drive solutions

20 … 20,000 A_{DC}
0 … 1,500 V_{DC}
230 … 1,200 V_{AC}
IP21 – IP54

- Individually adaptable to customer requirements
- User-defined accessories like external PLC or automation systems can be included
- High power solutions in 6- and 12-pulse up to 20,000 A, 1,500 V
- In accordance to usual standards
- Individually factory load tested
- Detailed documentation

DCS800-E series

Pre-assembled drive-kits

20 … 2,000 A_{DC}
0 … 700 V_{DC}
230 … 600 V_{AC}
IP00

- DCS800 module with all necessary accessories mounted and fully cabled on a panel
- Very fast installation and commissioning
- Squeezes shut-down-times in revamp projects to a minimum
- Fits into Rittal cabinets
- Compact version up to 450 A and Vario version up to 2,000 A

DCS800-R Rebuild Kit

Digital control-kit for existing powerstacks

20 … 20,000 A_{DC}
0 … 1,160 V_{DC}
230 … 1,200 V_{AC}
IP00

- Proven long life components are re-used, such as power stacks, (main) contactors, cabinets and cabling / busbars, cooling systems
- Use of up-to-date communication facilities
- Increase of production and quality
- Very cost-effective solution
- Open Rebuild Kits for nearly all existing DC drives
- tailor-made solutions for…
  - BBC PxD
  - BBC SZxD
  - ASEA TYRAK
  - other manufacturers

ABB Automation Products GmbH
Wallstadter Straße 59
68526 Ladenburg • Germany
Tel: +49 (0) 62 03-71-0
Fax:+49 (0) 62 03-71-7609
www.abb.com/motors&drives

Ident. No.: 3ADW 000 199 R0401 Rev D
01_2009

199R0401A9050000