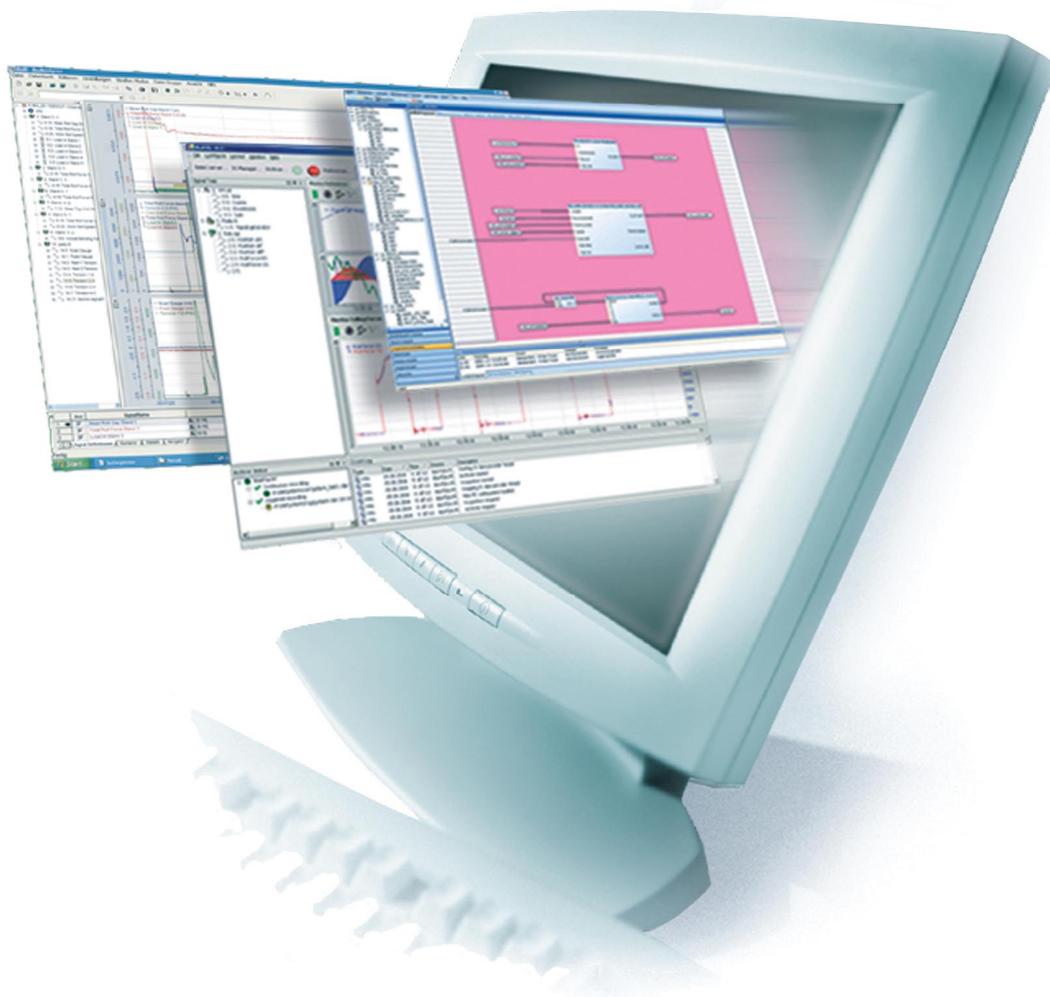


ibaLogic-V4



Manual

Issue 4.2.4

Measurement and Automation Systems



Manufacturer

iba AG
Koenigswarterstr. 44
90762 Fuerth
Germany

Contacts

Main office +49 911 97282-0
Fax +49 911 97282-33
Support +49 911 97282-14
Engineering +49 911 97282-13
E-Mail iba@iba-ag.com
Web www.iba-ag.com

This manual must not be circulated or copied, or its contents utilized and disseminated, without our express written permission. Any breach or infringement of this provision will result in liability for damages.

©iba AG 2013, All Rights Reserved

The content of this publication has been checked for compliance with the described hardware and software. Nevertheless, deviations cannot be excluded completely so that the full compliance is not guaranteed. However, the information in this publication is updated regularly. Required corrections are contained in the following regulations or can be downloaded on the Internet.

The current version is available for download on our web site <http://www.iba-ag.com>.

Protection note

Windows® is a label and registered trademark of the Microsoft Corporation. Other product and company names mentioned in this manual can be labels or registered trademarks of the corresponding owners.

Issue	Date	Revision	Author	Version SW
4.2.4	19.02.2013	Update Software	KF	4.2.4

Table of Contents

1	About this manual	11
1.1	Target group.....	11
1.2	Notations.....	11
1.3	Used symbols.....	12
2	Introduction	13
2.1	Identification.....	13
2.2	Proper Use.....	13
2.3	Release Notes	13
2.3.1	Change Log File.....	13
3	Software Installation	14
3.1	System Requirements.....	14
3.1.1	Hardware	14
3.1.2	Software.....	15
3.2	License Activation	16
3.3	Software Installation.....	17
3.3.1	Prerequisite.....	17
3.3.2	Procedure	17
3.3.3	Software required.....	18
3.3.4	System requirements	19
3.3.5	Choose components	20
3.3.6	Choose Installation Location.....	21
3.3.7	Select SQL server	21
3.3.8	Complete ibaLogic installation	23
4	ibaLogic Software	24
4.1	Introduction	24
4.2	Areas of Application	25
4.3	The ibaLogic Components	27
4.3.1	Runtime system (PMAC).....	28
4.3.2	ibaLogic Server.....	28
4.3.3	ibaLogic Client	28
4.3.4	OPC Server.....	28
4.4	Multi-client Operation and other System Configurations.....	29
4.5	Operating and Processing Modes.....	31
4.6	Structure of an ibaLogic application	32
4.6.1	Task / Program Properties.....	32
4.6.2	Program Elements	33
4.6.2.1	Function blocks.....	33

4.6.2.2	Graphics Programming	34
4.6.2.3	Comments.....	34
4.6.2.4	Data types available.....	34
4.6.2.5	Integrated measurement using ibaPDA Express.....	35
4.6.2.6	Measured value storage	35
4.7	Connectivity	36
5	ibaLogic Server.....	37
5.1	Functional overview of the ibaLogic Server	37
5.2	Start ibaLogic Server.....	38
5.3	User Interface – ibaLogic Server	40
5.4	ibaLogic Server Setting.....	41
5.4.1	Configuring the Client port	41
5.4.2	Configuring the Database Connections	42
5.4.2.1	Connect database.....	42
5.4.2.2	Configuring the Database Interface	44
5.4.2.3	Select SQL server.....	45
5.4.2.4	Manage Database scripts	46
5.4.3	Options.....	47
5.4.3.1	Activate Autostart Server	47
5.4.3.2	Configure General ibaLogic Server Options.....	49
5.4.3.3	Settings for the Local PMAC.....	50
5.4.3.4	Language	52
5.4.4	Status bar.....	53
6	Programming Environment – ibaLogic Client.....	54
6.1	Start ibaLogic Client.....	54
6.2	User Interface of Programming Environment – Editor	55
6.2.1	Menu Bar	55
6.2.2	Toolbar	55
6.2.3	Navigation Area.....	56
6.2.3.1	Switch Views in Workspace Explorer	57
6.2.3.2	Instances.....	58
6.2.3.3	Definitions	59
6.2.3.4	Hierarchy.....	60
6.2.3.5	Evaluation Order	60
6.2.4	Program Designer.....	62
6.2.5	Arrangement of the Tabs and Programming Windows.....	63
6.2.5.1	Arrange tabs	63
6.2.5.2	Arrange programming windows	63
6.2.5.3	Navigating in the Program Designer.....	65
6.2.6	Synchronize Access (<Read write>/<Read only> buttons).....	68
6.2.7	Events Window	68
6.2.7.1	Local Events	69
6.2.7.2	Server Events	69
6.2.7.3	All Events	69
6.2.7.4	Console View	69

6.3	Workspace	70
6.3.1	Create Workspace	70
6.3.2	Open workspace	71
6.3.3	Close Opened Workspace	71
6.3.4	Remove Workspace from the Database	72
6.4	Workspace Projects	73
6.4.1	Create Project	73
6.4.2	Set Project as Active	74
6.4.3	Load Project in the Program Designer	75
6.4.4	Edit Project Properties	75
6.4.5	Remove Project.....	75
6.5	Tasks/Programs	76
6.5.1	Create Tasks / Programs.....	76
6.5.2	Open Tasks/Program	77
6.5.3	Change Task / Program Properties	77
6.5.4	Remove Task / Program.....	78
6.5.5	Import / Export Programs	78
6.6	Configure Inputs and Outputs	80
6.6.1	Create Signals.....	81
6.6.1.1	Define Group	81
6.6.2	Define Signals.....	82
6.6.3	Edit Existing Signals.....	84
6.6.4	Remove Signals.....	84
6.6.5	Export / Import Signals.....	86
6.6.6	Using Signals in the Program.....	88
6.6.7	Remove Signals in the Program	89
7	Program Creation.....	90
7.1	Blocks	90
7.1.1	Using Blocks	91
7.1.2	Create User Blocks	92
7.1.2.1	In the Program.....	92
7.1.2.2	Under the project	92
7.1.2.3	In the Global Library	92
7.1.3	Managing Blocks.....	93
7.1.4	Exporting Blocks	93
7.1.5	Importing Blocks.....	94
7.1.6	Removing Blocks	95
7.2	Standard Blocks	96
7.3	Complex Function Blocks.....	96
7.3.1	DAT_FILE_WRITE (DFW Function Block).....	96
7.3.1.1	Function Block Edit DFW.....	96
7.3.1.2	"General Configuration" Sub-tab	98
7.3.1.3	Sub-tab "Signal configuration"	102
7.3.1.4	Generate Storage Structure	104

7.3.2	TCPIP_SENDRECV	105
7.3.2.1	Inputs	106
7.3.2.2	Outputs.....	107
7.3.3	PIDT1_CONTROL	107
7.3.3.1	Inputs	109
7.3.3.2	Outputs.....	109
7.3.3.3	Details / Signal trends	110
7.3.3.4	P component: (Parameter: KP, EN_P)	111
7.3.3.5	I component: (Parameters KP, TN, SET, SV, HI and EN_I).....	112
7.3.3.6	DT1 component: (Parameters KV,T1 and EN_D)	113
7.3.3.7	PIDT1 component – Total response	115
7.3.4	RAMP.....	116
7.3.4.1	Inputs	117
7.3.4.2	Outputs.....	117
7.3.4.3	Example	118
7.3.5	FUZZY_CONTROLLER.....	120
7.3.5.1	Inputs	121
7.3.5.2	Outputs.....	121
7.4	User-specific Function Blocks	123
7.4.1	Function Blocks.....	123
7.4.1.1	General Settings	124
7.4.2	Structured Text Editor.....	127
7.4.2.1	IntelliSense	128
7.4.2.2	Syntax Description of Structured Text	128
7.4.2.3	Operators	129
7.4.2.4	Statements.....	129
7.4.2.5	Constants	131
7.4.2.6	Strings	132
7.4.3	Macro block.....	133
7.4.3.1	Creating a Macro Block.....	133
7.4.3.2	Opening a Macro	134
7.4.3.3	Combining existing components into a Macro Block	134
7.4.3.4	Expanding a Macro Block	135
7.4.4	Creating your own DLLs	136
7.4.4.1	Source Files and Descriptions Required	137
7.4.4.2	Requirements and Notes	137
7.4.4.3	Integrating the DLL into ibaLogic	137
7.5	Data types	139
7.5.1	Define Data Type	139
7.5.1.1	Under the project	141
7.5.1.2	In the global library.....	141
7.5.1.3	When creating a Function Block	141
7.5.2	Modify Data Type	142
7.5.3	Delete Data Type	142
7.5.4	Manage Data Type.....	142
7.5.5	Export Data Type	143
7.5.6	Import Data Type.....	144
7.5.7	Use Data Type	144

7.5.7.1	During the Creation of a Function Block.....	144
7.5.7.2	During the Creation of a Structure Data Type.....	144
7.5.8	User-defined Data Types	144
7.5.8.1	DIRECT DERIVED TYPE Group.....	145
7.5.8.2	SUBRANGE TYPE Group.....	145
7.5.8.3	STRING DERIVED TYPE Group	145
7.5.8.4	ENUM TYPE Group.....	146
7.5.8.5	ARRAY TYPE Group.....	148
7.5.8.6	STRUCT TYPE Group.....	149
8	Program Elements	151
8.1	Create Program Element	151
8.2	Mark Program Elements	151
8.3	Move Program Element	152
8.4	Align Program Elements along an Edge	152
8.5	Copy Program Element.....	153
8.6	Delete Program Element.....	153
8.7	Generate Input / Output Variables.....	153
8.8	Graphical Connections.....	154
8.8.1	Direct Connectors	154
8.8.1.1	Types of connection lines.....	154
8.8.1.2	Create Direct Connector.....	154
8.8.1.3	Modify Direct Connectors	155
8.8.2	Intra-Page Connectors	155
8.8.2.1	Create Intra-Page Connectors	155
8.8.2.2	Modify IPC Names.....	156
8.8.2.3	Track IPC.....	156
8.8.3	Off-Task Connectors	157
8.8.3.1	Create Off-Task Connectors.....	157
8.8.3.2	Rename OTC	159
8.8.3.3	Track OTCs	160
8.8.3.4	List of all OTCs	161
8.8.3.5	Display.....	161
8.9	Converters, splitters, joiners.....	162
8.9.1	Converter	162
8.9.2	Splitter	163
8.9.3	Joiner	163
8.10	Comments.....	164
9	PMAC Runtime System	165
9.1	Overview of Online and Offline Modes.....	165
9.2	Start Runtime System	165
9.3	Stop the Runtime system	166
9.4	Runtime System – Autostart.....	167
9.4.1	Save program on the PMAC	167

9.4.2	Delete Program on the PMAC	168
9.5	Connect/disconnect	169
10	Platforms	171
10.1	Configuring the Platform	172
10.2	Selecting the Platform.....	174
11	IO Configuration	175
11.1	Resources.....	176
11.1.1	Hardware Resources	178
11.1.2	Software Resources.....	179
11.1.3	Global System Variables.....	179
11.2	Hardware Configuration	180
11.2.1	General Settings	180
11.2.2	Card Settings	181
11.3	Signal assignment.....	182
11.3.1	Method as seen from the hardware	182
11.3.1.1	Example: Assignment of all signals of a module of an ibaFOB-io-S card	183
11.3.1.2	Example: Assignment of individual signals of an ibaFOB-4i-S or ibaFOB-4o-S card	185
11.3.1.3	Change Signal and Group Names.....	186
11.3.2	Procedure as seen from the program	186
11.3.2.1	Example: Signals of an ibaFOB-4io-S card (complete module)	186
11.3.3	Modify Signal Assignment.....	188
11.3.4	Using externally defined signal names	188
11.4	PCI Interfaces (Windows PC)	190
11.4.1	Connection to the "iba World"	190
11.4.1.1	Card Settings	190
11.4.1.2	Link Settings	191
11.4.2	Buffered Mode	192
11.4.2.1	Applications.....	192
11.4.2.2	Input Resources.....	193
11.4.2.3	Output Resources.....	194
11.4.3	ibaLogic as Profibus Slave.....	195
11.4.3.1	Card Settings	195
11.4.3.2	Settings for bus interface 0/1	196
11.4.4	ibaLogic as Profibus Master.....	196
11.4.4.1	Brief Description.....	197
11.4.4.2	Card Settings	197
11.4.4.3	Configuration.....	198
11.4.4.4	Peculiarities with signal assignment	198
11.4.5	SIMADYN D / SIMATIC TDC Connection	199
11.4.5.1	Card settings.....	200
11.4.5.2	Link settings	200
11.4.5.3	Communication Settings	201
11.4.6	Reflective Memory	201
11.4.6.1	Brief Description.....	202
11.4.6.2	Card Settings	202

11.4.6.3	Configuration	202
11.4.6.4	File	203
11.4.6.5	Flow of Setting Parameters	204
11.5	ibaPADU-S-IT Platform	204
11.5.1	Settings	205
11.6	TCP/IP Communication.....	206
11.6.1	TCP/IP Connection Settings	206
11.7	OPC Communication	207
11.7.1	OPC Server.....	207
11.7.2	Setting the OPC Variable Parameters.....	209
12	Database Management	210
12.1	Backup Database.....	210
12.1.1	Manual Database Backup	210
12.1.2	Automatic Database Backup.....	211
12.2	Restore Database	214
12.3	Reset Database	216
13	Program Analysis, Debugging and Time behavior.....	217
13.1	ibaPDA Express	217
13.1.1	Controlling the Signal Display	218
13.1.2	Select Signals	218
13.1.3	Move signal.....	219
13.1.4	Mark the signals with color.....	220
13.1.5	Remove Signal from the Display.....	220
13.1.6	Remove Graphs from the Display	220
13.1.7	Scale Axes	221
13.1.7.1	Auto scaling	221
13.1.7.2	Scaling with the mouse.....	221
13.1.7.3	Scaling using the display settings	222
13.1.8	Move Scales	222
13.1.9	Zoom Function	223
13.1.9.1	Zooming in (Enlarge).....	223
13.1.9.2	Zoom out (Reduce).....	223
13.1.10	Trend graph Properties	224
13.1.10.1	Miscellaneous.....	225
13.1.10.2	Colors	225
13.1.10.3	Fonts.....	226
13.1.10.4	Signals.....	226
13.1.10.5	X-axis.....	226
13.1.10.6	Y-axis.....	227
13.1.10.7	Scientific notation	227
13.1.10.8	Scaling mode.....	228
13.1.11	Extended Functionality.....	228
13.2	Time behavior	230
13.2.1	Evaluation time	231

13.2.2	Turbo mode.....	231
13.2.3	Messung	232
13.2.4	Soft PLC.....	232
13.2.5	Time considerations with multiple tasks.....	233
13.2.6	Worst-case considerations.....	234
13.2.7	Explanation of the case above.....	234
13.2.8	Task evaluation with time shift	235
13.3	Debugging.....	237
13.3.1	Program errors.....	237
13.3.1.1	Errors in user-defined function blocks	237
13.3.1.2	Division by 0.....	237
13.3.1.3	Incorrect signal trends.....	237
13.3.1.4	Evaluation sequence.....	237
13.3.2	Compilation errors.....	238
13.4	Performance Limits	240
13.4.1	Example	240
14	Programming rules.....	242
14.1	Approach for the solution.....	242
15	Uninstall ibaLogic.....	245
16	Practice Examples	248
16.1	First Steps - Sample Project	248
16.1.1	Sample Exercise Part 1	249
16.1.1.1	Task Description	249
16.1.1.2	Start ibaLogic Server and ibaLogic Client.....	250
16.1.1.3	Create a New Project.....	251
16.1.1.4	Placing the Test Tools	252
16.1.1.5	Placing the evaluation blocks	253
16.1.1.6	Connecting the selector block with the test tools.....	254
16.1.1.7	Configuring the slider and generator	255
16.1.1.8	Switch the partial connections online.....	256
16.1.1.9	Testing the switch and selector	257
16.1.1.10	Connecting the adder.....	258
16.1.1.11	Create an OTC to illustrate the result.....	258
16.1.1.12	Analysis of the circuit	259
16.1.2	Sample Exercise Part 2	260
16.1.2.1	Program analysis using the ibaPDA Express	260
16.1.3	Sample Exercise Part 3	261
16.1.3.1	Procedure.....	261
16.1.3.2	Remark.....	262
16.1.4	Sample Exercise Part 4	262
16.1.4.1	Procedure.....	263
16.1.4.2	Remark.....	264
16.1.4.3	Result.....	265
16.1.4.4	Remarks.....	266
16.2	DAT_FILE_WRITE Sample Project.....	267

16.2.1	DAT_FILE_WRITE in "Unbuffered" Mode	267
16.2.1.1	Step 1: Configure the DFW block	267
16.2.1.2	Step 2: Connection of the DFW	268
16.2.1.3	Step 3: Create other measure signals	269
16.2.1.4	Step 4: Starting the recording	270
16.2.1.5	Alternative: Programming Joiner in ST	270
16.2.2	DAT_FILE_WRITE in "Buffered Mode"	271
16.2.2.1	Step 1: Configuration of the buffered inputs	272
16.2.2.2	Step 2: Set the parameters of the DFW module, "General Configuration"	273
16.2.2.3	Step 3: Accept the buffered input signals	274
16.2.2.4	Step 4: Transfer the data to DAT_FILE_WRITE	275
16.2.2.5	Step 5: Wiring (Connecting) the remaining inputs	275
16.2.2.6	Step 6: Starting the recording	276
17	Naming conventions	277
18	Data types	278
18.1	Standard data types	278
18.2	Derived data types	278
18.3	Generic data types	279
19	Standard Function Blocks	280
19.1	Table interpretation	280
19.2	Data types	280
19.3	Block type with function diagram display	281
19.4	Analytical Functions	282
19.5	Arithmetical Functions	284
19.5.1	General	284
19.5.2	Logarithmic	284
19.5.3	Trigonometric	285
19.5.4	Miscellaneous	286
19.6	Bistable	288
19.7	Bit String	289
19.7.1	Bit shift	289
19.7.2	Bitwise_Boolean	290
19.8	Character String	291
19.9	Communication	292
19.10	Comparison	293
19.11	Counter	295
19.12	Edge Detection	296
19.13	Register	297
19.14	Selection	298
19.15	Signal Processing	300
19.16	Specials	301

19.17	Timer	304
19.18	Type Conversion	306
19.18.1	Limiting Converter	308
19.18.2	Scaling Converter	311
19.18.3	Standard Converter	313
20	Error Codes	314
20.1	DAT_FILE_WRITE Error Codes	314
20.2	TCPIP_SENDRECV Error Codes	314
21	Characteristics of TCP/IP	318
21.1	Number of TCP/IP connections possible	318
21.2	Delayed Acknowledge Problem	318
22	Key Combinations	320
22.1	Client	320
22.2	Mouse Functions in the Programming Field	320
22.3	ibaPDA Express	321
23	Character tables	322
24	Index of Abbreviations	324
25	Classified Index	326
26	Support and contact	330

1 About this manual

This documentation describes the function, the design and the application of the software ibaLogic-V4.

1.1 Target group

This manual addresses in particular the qualified professionals who are familiar with handling electrical and electronic modules as well as communication and measurement technology. A person is regarded as professional if he/she is capable of assessing safety and recognizing possible consequences and risks on the basis of his/her specialist training, knowledge and experience and knowledge of the standard regulations.

1.2 Notations

In this manual the following notations are used:

Action	Notation
Menu command	Menu "Logic diagram"
Calling the menu command	„Step 1 – Step 2 – Step 3 – Step x” Example: Select the menu "Logic diagram - Add - New function block".
Keys	<Key name> Example: <Alt>; <F1>
Press the keys simultaneously	<Key name> + <Key name> Example: <Alt> + <Ctrl>
Buttons	<Key name> Example: <OK>; <Cancel>
File names, paths	"Filename", "Path" Example: "Test.doc"

1.3 Used symbols

If safety instructions or other notes are used in this manual, they mean:

DANGER

The non-observance of this safety information may result in an imminent risk of death or severe injury:

- From an electric shock!
 - Due to the improper handling of software products which are coupled to input and output procedures with control function!
-

WARNING

The non-observance of this safety information may result in a potential risk of death or severe injury!

CAUTION

The non-observance of this safety information may result in a potential risk of injury or material damage!



Note

A note specifies special requirements or actions to be observed.



Important note

Note if some special features must be observed, for example exceptions from the rule.



Tip

Tip or example as a helpful note or insider tip to make the work a little bit easier.



Other documentation

Reference to additional documentation or further reading.

2 Introduction

2.1 Identification

PAC (Soft PLC) and signal manager "ibaLogic-V4".

2.2 Proper Use

The product / system is used for the measurement and control of technical plants and systems.

ibaLogic is not designed for safety-related systems.

Any other or extended use of the product / system is deemed to be improper, and hence, misuse. In this case, the safety and protection of the product / system may get impaired or compromised. iba AG is not liable for any loss or damage resulting from such misuse.

 DANGER**Danger by enabling functions or other services!**

Possibility of human injuries and damage to machinery by enabling functions and other services (PMAC, OPC ...), which have direct impact on the response of the system.

Secure the system while working on it! Follow the safety regulations applicable!

2.3 Release Notes

2.3.1 Change Log File

A change log file (changelog.htm) for your software is available on the installation media. It contains, among others, important information on the following topics:

- New functions
- Error corrections

3 Software Installation

3.1 System Requirements

3.1.1 Hardware

The hardware requirements are listed in the following table.

	Minimum requirement	Recommended or higher
CPU speed	1600 MHz	2000 MHz
Number of CPUs	1	2
RAM	768 MByte	2048 MByte
Screen resolution	1024 x 768	1280 x 1024

The minimum memory requirement is about 650 MB. An SQL server express database can grow in size up to 4 GB. Keep sufficient free memory space for your requirements. For more information, please refer to "Performance Limits, Page 240".



Note

Installation is possible if the minimum requirements are not met. However, performance limitations may arise.

3.1.2 Software

One of the following Operating Systems must be pre-installed:

- Windows XP Professional SP3
- Windows 2003 server
- Windows 7 SP1 32bit



Note

Administrator rights are required for both the installation and operation of the ibaLogic Server and Client.

The software packages listed below are part of the CD supplied:

- Windows Installer 3.1
- MDAC 2.81
- .Net Framework 2.0 SP1
- MS SQL Server Express 2005 (9.0)
- OPC Core Components 2.0
- ibaWDM driver
- CB-USB Dongle driver
- Visual J# 2.0

The installation wizard checks whether the versions of various software packages are available. If any software packages are missing or are an older version, they are installed by the installation wizard or updated.

3.2 License Activation

The dongle is already customized at the time of delivery. The customer dongle generates a virtual key in the system that unlocks or activates various functions.

⚠ CAUTION

Danger due to switching off the runtime system PMAC after removing the dongle!

The system cannot be commissioned without the dongle having the associated license. The license determines the unlocking or activation of functions. Thus, leave the dongle inserted during the entire operation!

If the dongle is removed during the operation, the PMAC (Programmable Measurement and Automation Controller) switches off following repeated warnings (ca. 5 min after the first warning).

The PMAC does not start without a dongle. Instead, an alternative demo version of the PMAC can be enabled in the ibaLogic Server options (see "Settings for the Local PMAC, Page 49"). If the PMAC is started manually or automatically by the ibaLogic Server, a warning message occurs saying that no dongle is inserted.



The demo version does not support hardware access, instead, several devices are simulated. Here, if possible, the outputs used are directly reconnected to the inputs (not possible e.g. with buffered inputs).

Several function blocks are disabled, i.e. they are configurable in the plan, but are not calculated, this includes among others: TCP/IP_SendRecv, DatFileWrite, User-DLLs.



figure 1: Dongle

Procedure

- ➔ Connect the dongle to a USB interface.

After starting the server and the client, the following message appears in the console view: "Online Server: DriverStatus: Driver running for Dongle Vxxxxxx".

**Note**

This message only occurs, if a project has been started.

3.3 Software Installation

Follow the instructions of the installation wizard to install the ibaLogic software.

3.3.1 Prerequisite

- Your system meets the requirements of the hardware and software.

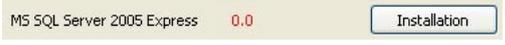
3.3.2 Procedure

- ➔ Double click on the file "Setup-4.x.xx.exe".

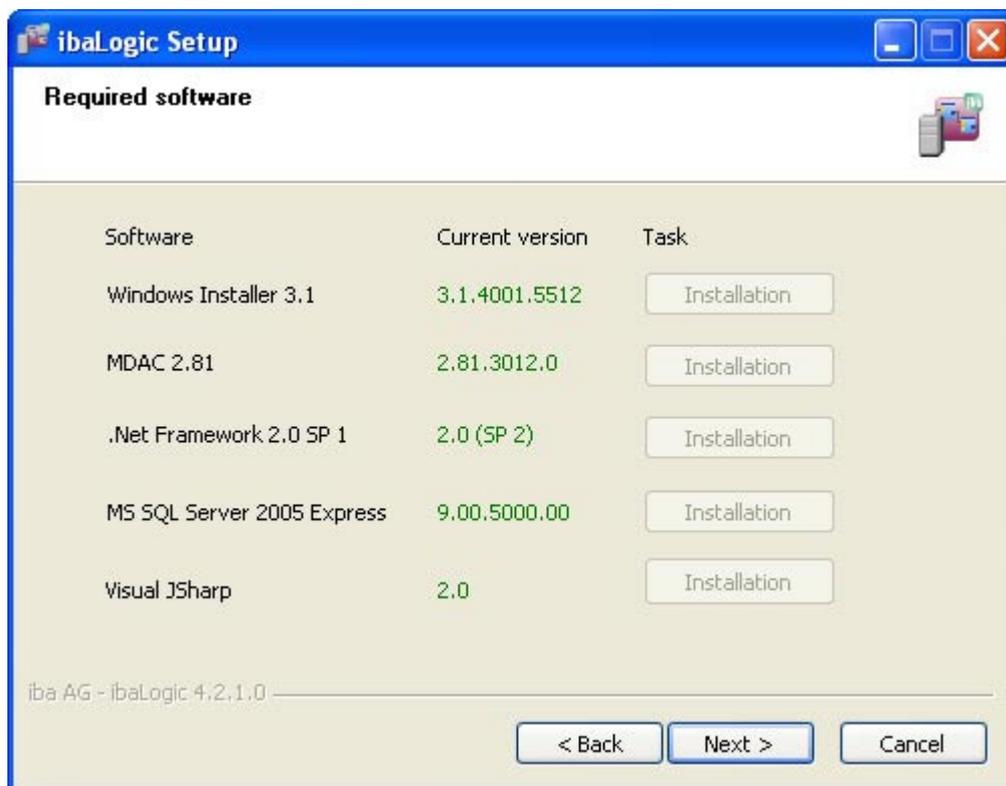


3.3.3 Software required

The components required and their versions are specified under "Software".

Presentation	Explanation
Version number green 	Software is installed with adequate functionality.
Version number red 	Software is not installed or inadequate. The associated <Installation> button is enabled.

- ➔ Install or update the software component(s) by clicking on the <Installation> button that is enabled.



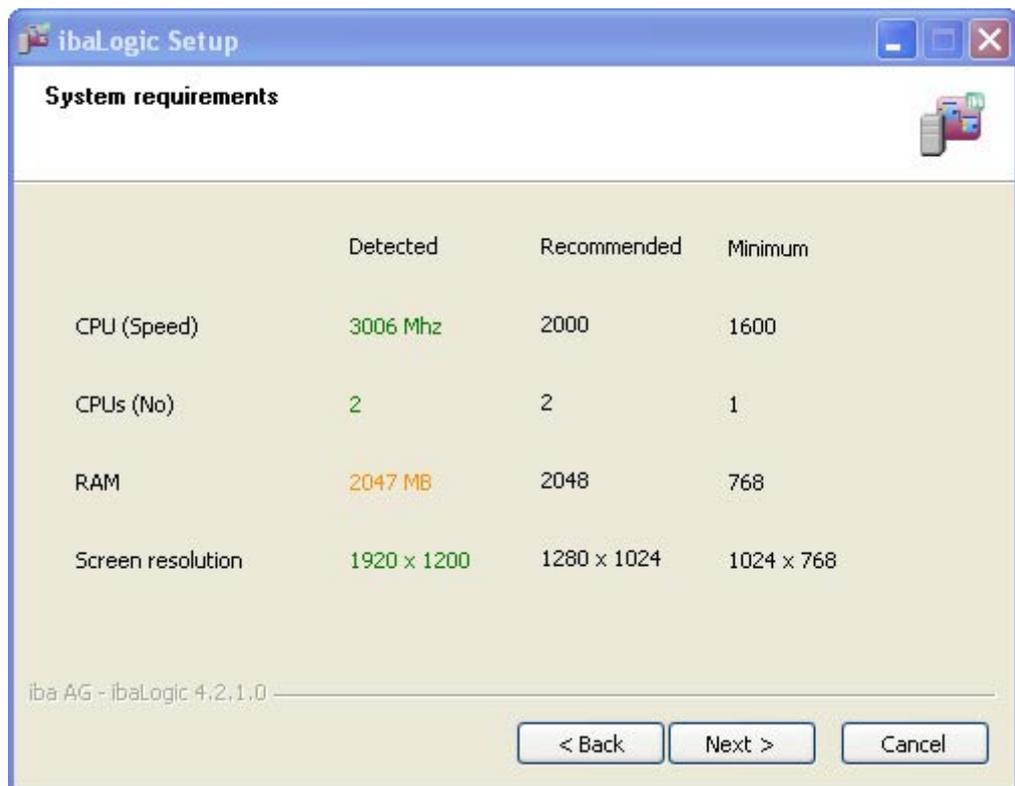
- ➔ If required, confirm this in the dialog window that appears. If all software components have been installed or updated, click on <Next>.

3.3.4 System requirements

The system requirements are checked prior to installation of the software components.

Presentation	Explanation
Green	The recommended requirements have been met.
Orange	The minimum requirements have been met.
Red	The minimum requirements have not been met.

- ➡ If the system requirements meet the minimum requirements, continue the installation.

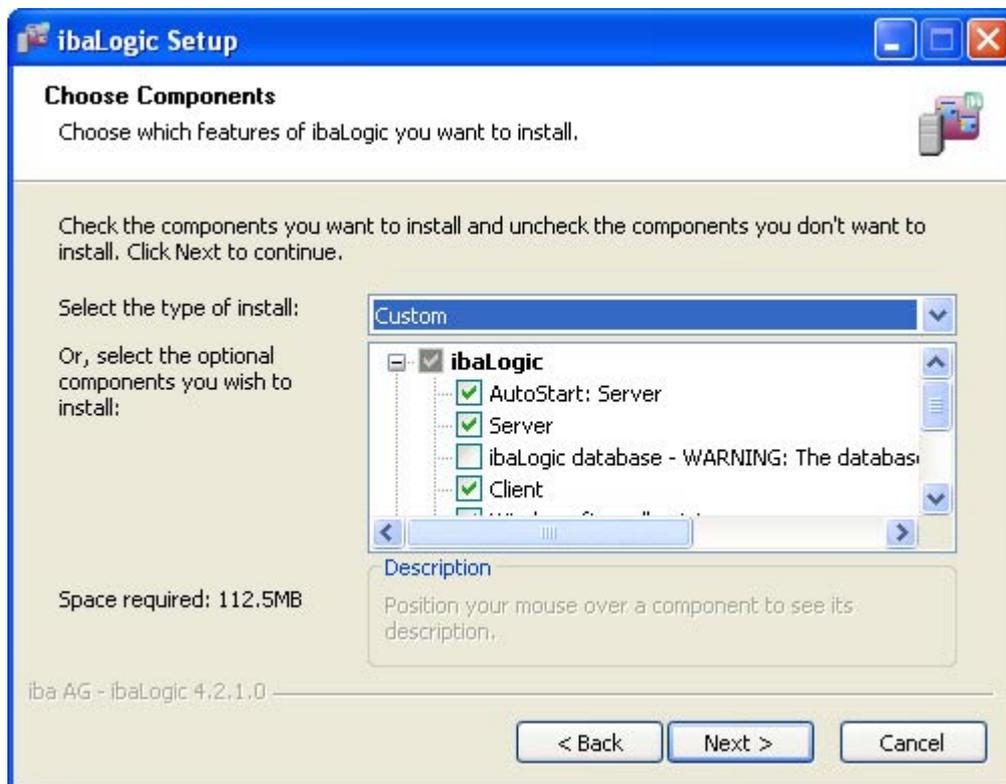


3.3.5 Choose components

You can use the "Installation type" selection field to choose components prior to the installation.

Installation type	Explanation
Complete	All components are installed.
Only server	Only the ibaLogic Server components are installed.
Only client	Only the ibaLogic Client is installed.
User defined	It is possible to choose the components. Only the components selected are installed.

- ➔ Define the ibaLogic components by selecting the installation type.



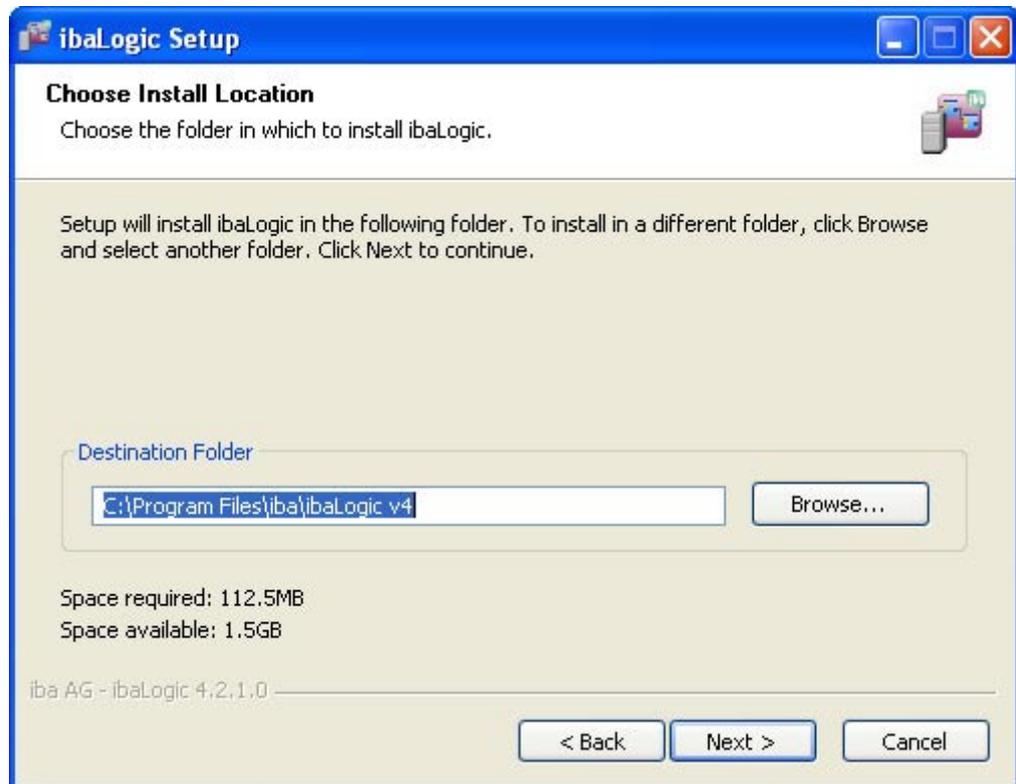
CAUTION

The "ibaLogic Database" option is not set in case of an update. When the option is set, the existing database including its projects are deleted in the course of installation. "The database already exists and will be overwritten" is displayed as a warning message.

3.3.6 Choose Installation Location

The ibaLogic folder structure (server, client etc.) is created in this folder. iba recommends using the default folder specification.

- ➔ Define the target folder.



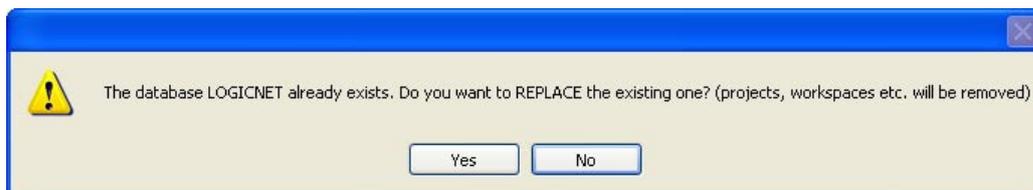
3.3.7 Select SQL server

If the "ibaLogic Database" check box has been enabled when installing, the installer searches for Microsoft SQL servers on the local computer during the installation process and offers you the "Select SQL server" dialog for selecting the database instance.

- ➔ Select the "<PC name>\IBA" default instance or an instance of your choice and quit the dialog window using the <OK> button.
The ibaLogic database selected is installed on the server selected.

**Note**

Any existing ibaLogic database can be overwritten after confirmation in the dialog window that appears. If you exit this dialog window using the <No> button, the existing database remains unchanged.



3.3.8 Complete ibaLogic installation

Complete ibaLogic installation

- ➔ Click on the <Finish> button to complete the installation.



4 ibaLogic Software

4.1 Introduction

iba AG has already specialized in the field of measured value acquisition in heavy industry plants for many years. The segment focus has been on plants for the production and processing steels and non-ferrous metals.

The programs for the acquisition¹ and analysis² of the data recorded are in use all across the globe today, and are deployed by all large suppliers of machinery and automation technology worldwide.

As a result of the wide-ranging options for the connection of the iba measurement technology to the most diversified automation technologies and generations, particularly even to the most prevalent field and drive buses, the need to connect these, at times, highly diverse worlds, developed rather quickly. From now on, "unidirectional flow" of measurement needs to become "bidirectional flow" for information exchange between various automation systems – this is typical for the upcoming market of modernization or revamping of automated systems that are already in existence.

In order to address this requirement, iba AG has already developed a freely programmable signal manager since 1995. The standard, IEC 61131-3, which had already been formulated during this period, for describing technical work flows with the help of graphics elements and easily embedded programming techniques, simplifies the descriptions of complex signal processes considerably.

The graphical mode of programming³ that has been derived from this standard, forms the basis of almost all automation systems today. As a result, graphics programming is compatible and portable to a large extent.

Features:

- Onlinechange
- Permanent project backup
- On-the-fly input check
- Visualization and trace tool (ibaPDAExpress)

¹ ibaPDA

² ibaAnalyzer

³ FBD is a graphical programming language, in which function blocks are interconnected with one another instead of a sequence of textual commands, as in the case of classical programming languages. Circuit diagrams of hardware development can be considered a model in this case. This representation of a program meets the requirements of developers of controller software, whose technical background is typically one of electrical engineering. The various function blocks are themselves often created using other PLC languages, . such as, for example, "structured text", and can be supplied as standard blocks by the manufacturer of the automation systems or written by the user himself or they can even be imported.

4.2 Areas of Application

ibaLogic is used for the following applications:

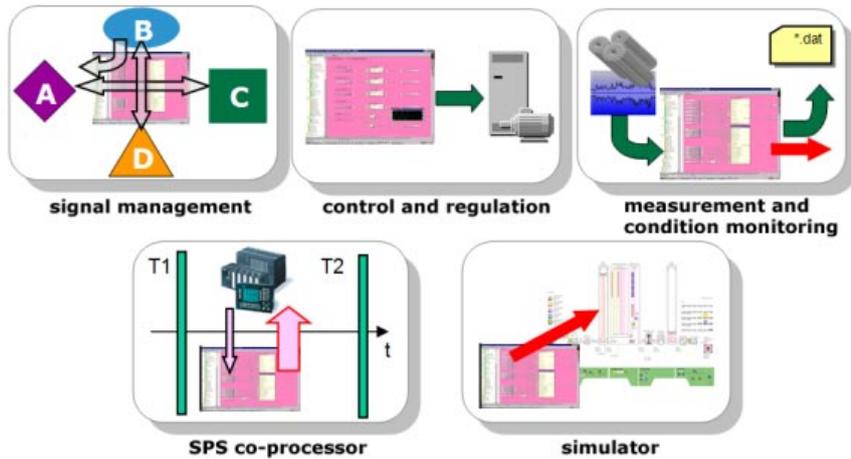


figure 2: Areas of Application

Signal management

You can establish links between the most diverse generations of automation systems from the largest variety of manufacturers with the help of iba connectivity.

Bi-directional data exchange enables communication with controllers that are otherwise incompatible.

SPS-co-processor

In this area of application, the ibaLogic plays the role of a co-processor.

The vertical green lines in the figure "Areas of application" represent the sampling time of the original automation equipment. Data is transmitted after one PLC clock pulse (T1) to the Soft PLC (ibaLogic). Complex calculations can be performed in real time and the results transmitted back before T2 using the PC processing power and the PC data formats available. Modernizations or revamps, too, can be implemented with the help of such methods: Open-loop and closed-loop control and regulations functions of the "old" PLC are taken over by the new ibaLogic automation system step by step.

Observation of measurements and status (Condition Monitoring)

Apart from the use as a signal manager, it was also desired to involve ibaLogic for complex measurement tasks, which would not have been possible using a standard PDA. Integrating a block for measured value sampling is one of the core functions of ibaLogic. You can achieve event-driven management of measurement tasks and save the values on various media using this "DAT_FILE_WRITE". You can then process and analyze the data using various iba tools, e. g. ibaAnalyzer, ibaDatCoordinator, etc.

Automation (Control & Regulation)

The graphical programming language described in the IEC 61131-3 standard forms the basis of ibaLogic. This language has been conceived particularly for programmable logic controllers (PLC). The developments in recent years have shown that the market for measurement and control systems is growing together increasingly. The logical consequence is that ibaLogic can obviously also be used for automation tasks as a full-fledged PLC.

If, in the process, the tasks of the Operating and Runtime system are handled by a PC, then one speaks of a PC-aided Soft PLC or PAC, in short (Programmable automation controller).

ibaLogic permits demarcation of the Runtime system of the PC into a secondary stand-alone intelligence system, ibaPADU-S-IT. In such a case you can shut down the PC without stopping the Runtime system. The PC is used as a development station in such cases, as in all other PLC systems.

Simulator

The simulator is an active one and an application that is programmed using ibaLogic. HMI visualization presents the operator interface and the result of the simulation. The standard OPC interface is used to establish the connections between ibaLogic simulation and the HMI.

4.3 The ibaLogic Components

ibaLogic is based on the server-client model. This architecture facilitates decentralized and multi-client operation.

ibaLogic is composed of the following components:

- Runtime system (PMAC)
- ibaLogic Server
- ibaLogic Client
- Database
- OPC Server

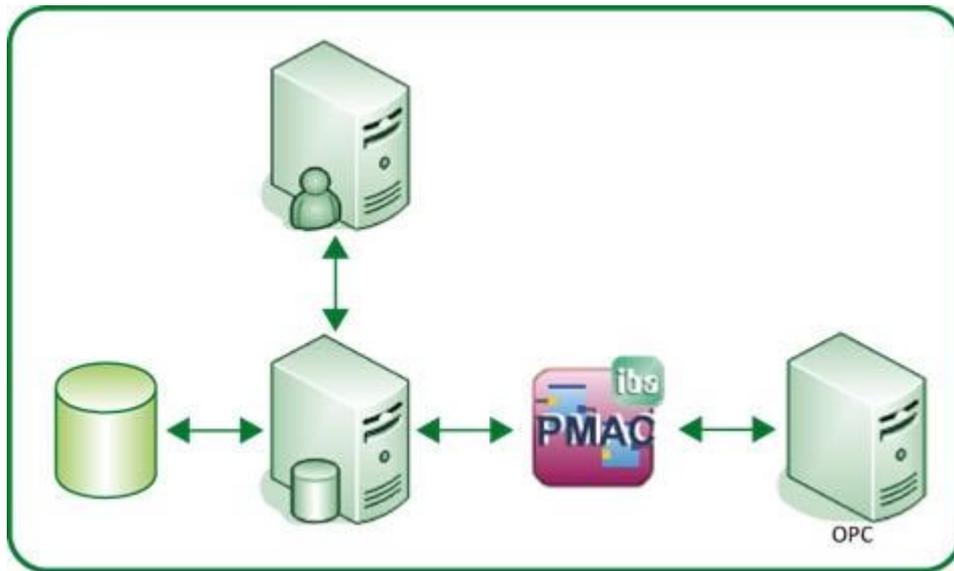
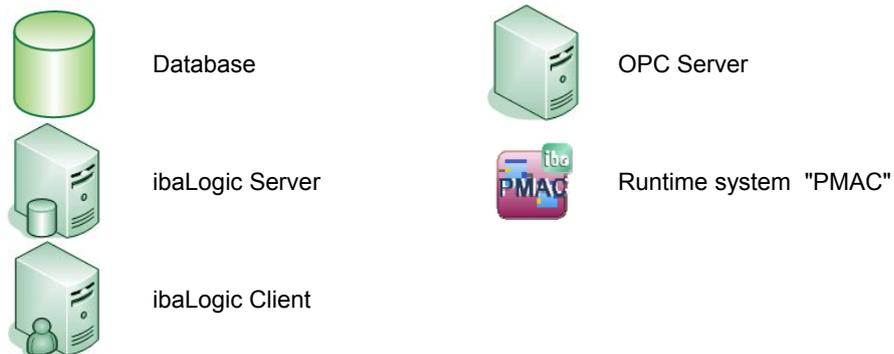


figure 3: ibaLogic Components



In the simplest case, both the components mentioned above are located in one computer. However, you can also run these components on separate computers.

4.3.1 Runtime system (PMAC)

By starting an ibaLogic project in the ibaLogic Client, it gets compiled and loaded in the "Programmable Measurement and Automation Controller" (PMAC). This Runtime system can be located on a Windows computer (as Windows service) or on a Windows CE-compatible device (ibaPADU-S-IT).

The Runtime system continuously returns values that are calculated currently to the client. These are displayed on-line in the graphical user interface of the project.

If a project has been transferred to the Runtime system and started, it is capable of running independently without the server / client running.

4.3.2 ibaLogic Server

ibaLogic is a database-based system. The ibaLogic Server takes over the management of the database and the communication between the ibaLogic Client and the Runtime system as the central manager.

All ibaLogic projects are managed by the ibaLogic Server in a database.

ibaLogic uses the unlicensed Microsoft SQL Express database. During installation, a Microsoft SQL Express server with the associated database is installed if it is not already present.

The ibaLogic Server dialog is also used to backup and restore ibaLogic applications. The backup of the database is saved in an external file.

If there are any modifications made to ibaLogic projects, these are automatically saved in the database. Specific saving during the customization of a project is omitted.

4.3.3 ibaLogic Client

The ibaLogic Client is the programming environment in which ibaLogic projects are programmed and configured. An ibaLogic Server must be connected for this purpose. This ibaLogic Server can be present on the same computer or in the network.

Over and above this, the ibaLogic Client controls loading, starting and stopping an ibaLogic project in the Runtime system with the help of the ibaLogic Server.

4.3.4 OPC Server

The OPC Server provides all variables, which have been declared as OPC visible, to the OPC Clients connected. In general, OPC Clients are HMI (Human Machine Interface) systems. By default, the OPC Server runs on the same machine as the ibaLogic Server, but it can also be explicitly started on other computers in the network, and is then connected directly with the PMAC via TCP/IP independently.

4.4 Multi-client Operation and other System Configurations

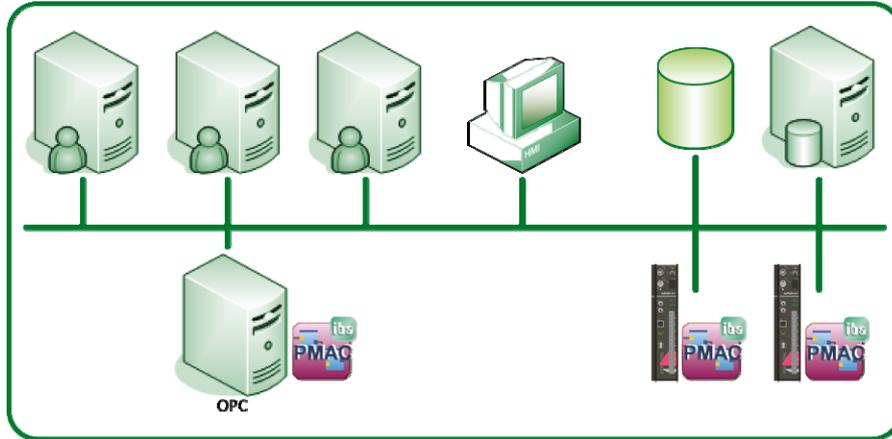
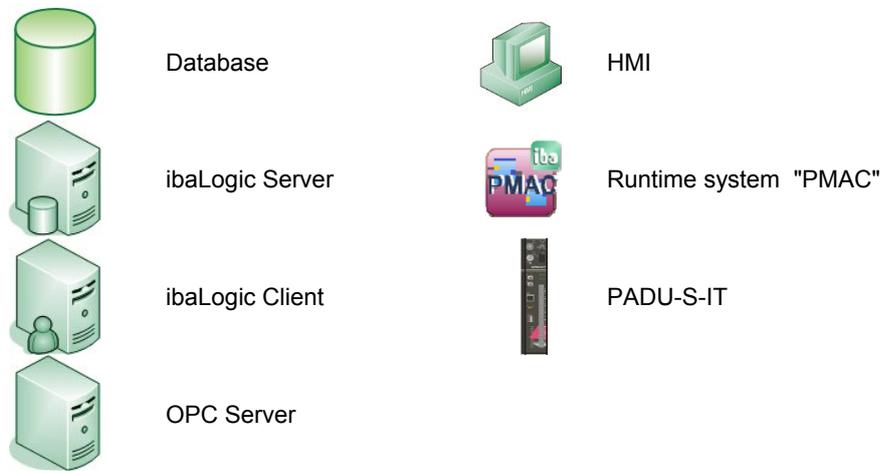


figure 4: Possible system configuration



In the simplest case, ibaLogic can run with all its components on a single Windows computer.

Alternatively, the ibaLogic components can also be distributed and run on different computers. There are 3 ibaLogic applications in the sample configuration illustrated above. One runs on a PC and 2 others on separate PADU-S-IT systems. The central server has a connection to one database in which all 3 projects are backed up. Only one ibaLogic workspace is always loaded on this server from the database, which can contain three projects corresponding to the three PMAcs.

It is possible to control and monitor the workspaces from different computers in multi-client mode of operation.



Important Note

Only one client at a time should carry out modifications regarding the projects of a server.

However, only one project / application as selected is always "active" in one workspace. You can only work with and monitor this active PMAC **online**.

The HMI system can be fed information via the OPC Server. No OPC Server can run on the PADU-S-IT. As a result, HMI data from and to the PADU-S-IT head units must run via this Windows computer.

4.5 Operating and Processing Modes

ibaLogic provides a range of operating modes in order to meet the various requirements of different applications. Several processing modes have been implemented since ibaLogic works not only as a soft PLC, but also as a signal manager, signal processor or simulator.

You can choose the following modes of operation in ibaLogic:

- Measurement
- Soft PLC

Set operating mode

Procedure

1. Select "Tools – I/O configurator" in the menu. The "I/O configurator" window is displayed.
2. You will find the operating mode options in the tab "Hardware configuration – General settings".
3. Activate the operating mode to be used under "General settings".
4. Finally, click on <Accept>.
5. If you wish to close the I/O configurator, click on <OK>.

You can choose the following processing modes in ibaLogic:

"Buffered mode" (= Packet transmission)

For explanations, please refer to "Time behavior, Page 230" and "Buffered Mode, Page 192".

Set processing modes

Procedure

1. In the tree on the left, mark the hardware whose processing modes you would like to set. The "Hardware configuration" tab for this hardware is displayed.
2. Activate the desired mode under "Connection settings".
3. Finally, click on <Accept>.
4. If you wish to close the I/O configurator, click on <OK>.



Note

For more information, please refer to "General Settings, Page 180".

4.6 Structure of an ibaLogic application

An ibaLogic project application consists of the following elements:

Workspace

- Project 1
 - Task 1 / Program 1
 - Task 2 / Program 2
 - Task n / Program n
- Project 2
 - Task 1 / Program 1
 - Task 2 / Program 2
 - Task n / Program n
- Project n

You can assign the programs with their properties (task interval etc.) to one project each. The projects, in turn, are organized in a workspace.

One project is assigned to one Runtime system / PMAC. Only one project is set as active within one workspace, i. e. only this active project can be started or stopped.



Note

Only one project can be active within a given application.

4.6.1 Task / Program Properties

According to IEC 61131-3, several programs can be assigned to one task. ibaLogic supports fixed assignment of one program to one task.

The following properties can be assigned to each task:

- Interval time
- Priority

The interval time determines the time slot in which the task is restarted. The minimum time slot for the interval time is 1 ms.

The priority setting determines the sequence in which the interval programs of a project are executed, starting with priority 0.



Note

The program properties "Interval time" and "Priority" are considerably significant for the project performance. For more detailed information on these program properties, please refer to "Time behavior, Page 230" and "Performance Limits, Page 240".

4.6.2 Program Elements

An ibaLogic program may consist of the following elements:

- Function Blocks
- Function blocks of the integrated standard library
- Function blocks created by the user with structured text
- Macro blocks created by the user
- DLL-based function blocks created by the user
- Linking elements
- Hardware input and output signals
- Comments

4.6.2.1 Function blocks

ibaLogic has a library of function blocks. This library contains standard function blocks in accordance with IEC 61131-3 and also supplementary function blocks.

You can combine these into a macro block to have a clearer program structure and provide encapsulation of various graphics subprograms.

You also have the option to create a separate block yourself that is required for a specialized solution to a problem.

For this purpose, ibaLogic provides the feature of creating a new function block with the help of structured text. The ST (Structured Text) code is visible to the user, who can modify it.

One variant of the self-created function block is creating your own DLLs (using a DLL framework provide by iba). The code is hidden with this option. The block created in this manner is available as a standard function block.



Other Documentation

For further information, please refer to the documentation on creating DLLs on the supplied CD "iba software and manuals".

4.6.2.2 Graphics Programming

The following elements are available to link the function blocks:

- Connection lines
- Intra-Page Connectors (IPC)
- Off-Task Connectors (OTC)
- Converter
- Splitter
- Joiner

You need to connect the function blocks with the help of connection lines for graphics programming. You can use intra-page connectors for better program structuring.

An intra-page connector merely represents a drawing simplification. In the process, the IPC replaces a connecting line. This is beneficial particularly when several objects on one page need to be connected with the same point or "long" connections are required across multiple pages.

Off-task connectors serve as program-independent connecting elements. These are required whenever you need communication between several programs.

Off-task connectors are also used for communication between ibaLogic and OPC Clients. This can be configured in the off-task connector.



Tip

For further information, also refer to "Graphical Connections, Page 154" and "Converters, splitters, joiners, Page 161".

4.6.2.3 Comments

You can use comments to structure and simplify the program description. These can be placed wherever desired or even "docked" to another element.

4.6.2.4 Data types available

ibaLogic supports all elementary and combined data types defined in the IEC 61131-3 standard (Exception: WSTRING).

4.6.2.5 Integrated measurement using ibaPDA Express

You can use the integrated ibaPDA Express tool for quick display of a signal waveform.

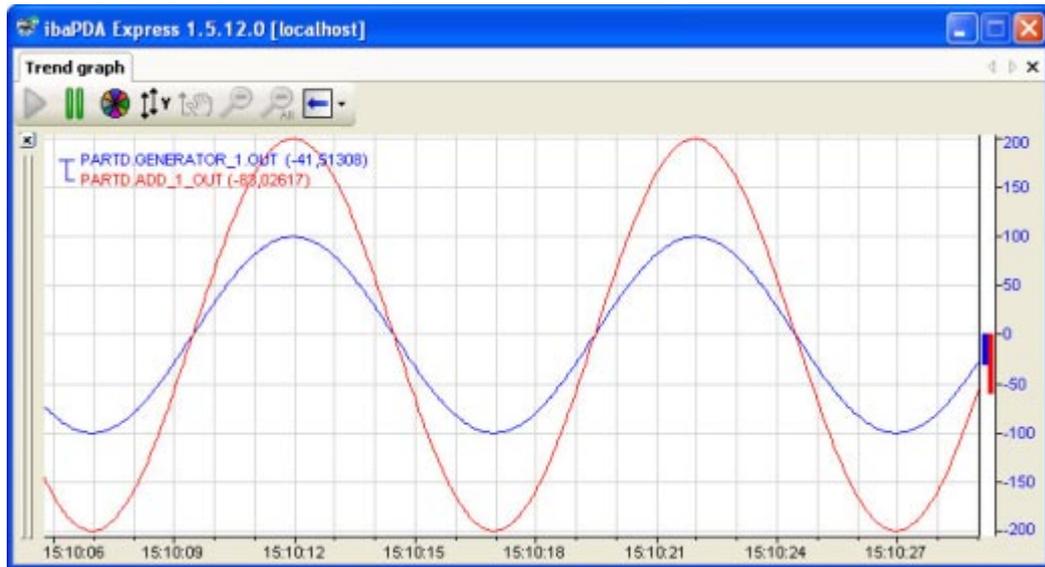


figure 5: Integrated measurement using ibaPDA Express

With the ALT key pressed, you can move the signals to the ibaPDA Express window using Drag & Drop and display them there.

ibaPDA Express does not save any data for long-term recording.

4.6.2.6 Measured value storage

You can store measured values using the licensed (rights-managed) function block, "DAT_FILE_WRITE". For more information, please refer to "DAT_FILE_WRITE (DFW Function Block), Page 96".



Tip

You can display and evaluate the signal measurements in the *.dat files created with the help of the user-friendly and comfortable analysis software, ibaAnalyzer.

5 ibaLogic Server

5.1 Functional overview of the ibaLogic Server

The ibaLogic Server is not only the central point of communication between ibaLogic Client and the PMAC, but it is also responsible for the management of the ibaLogic-V4 projects in the database. Similarly, in the background, it manages a connection to an active PMAC for loading / starting / stopping actions of the PMAC. This link is established via the ibaLogic Client.

Hence, the server can be divided in the following functions:

- ❑ Server operation
 - Start / Stop / Close
 - Database actions
 - Backup and restore
 - Resetting the entire current database
- ❑ Administrative settings
 - Set the port number for client connections
 - Set up connections to ibaLogic databases and their parameters
 - Configure auto start for the server and the local PMAC
 - Set up automatic backup of the current database
 - Set the general server options
 - Display status of / execute the database scripts (only for support purposes)

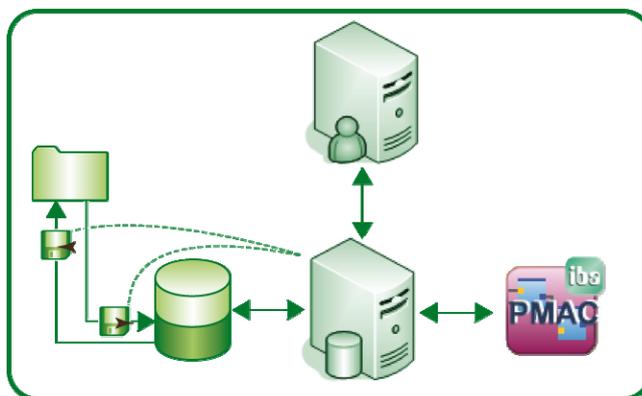
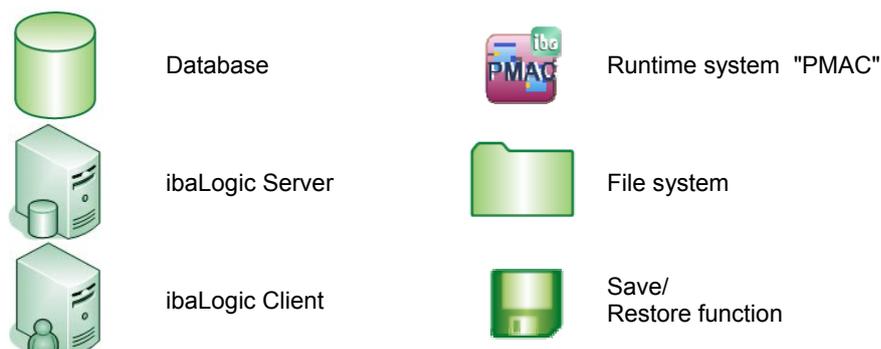


figure 7: Functional overview of the ibaLogic Server



5.2 Start ibaLogic Server

Requirement

You have the ibaLogic Server shortcut on the desktop.

Procedure

1. Double click on the shortcut "ibaLogic Sever" on the desktop.
The "ibaLogic Server" dialog box is displayed.



When opening the ibaLogic Server, it goes automatically to the start status. The following icons  are displayed in the info section.

- Click on the <Start> button to start the ibaLogic Server. You can also start the server via the menu "Server - Start".
The start / stop status is displayed in the server dialog box by an icon, a message and the active button.



5.3 User Interface – ibaLogic Server

The ibaLogic Server is used to:

- Configure the server
- Put it in the start / stop mode
- Configure and backup the database

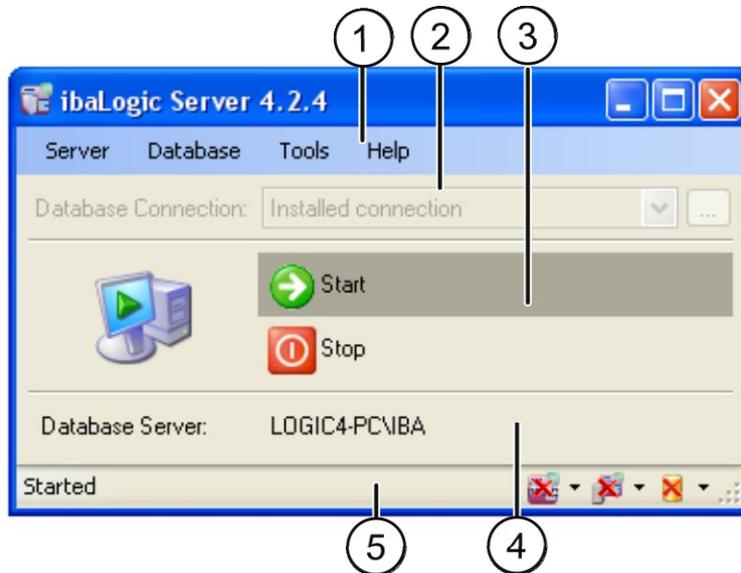


figure 8: ibaLogic Server - User interface

- | | | | |
|---|------------------------------------|---|-----------------------------|
| 1 | Menu bar | 4 | Current database connection |
| 2 | Setting up the database connection | 5 | Status bar |
| 3 | Start button / Stop button | | |

5.4 ibaLogic Server Setting

5.4.1 Configuring the Client port

The client port number serves as a link parameter for an ibaLogic Client.



Note

This setting should be changed only if a service that uses this port is being executed on the server computer. The same port must be configured in the client for connecting with this server. Enter the new port number also while selecting the link. Select the menu „File - Connect with Server...” of the client.

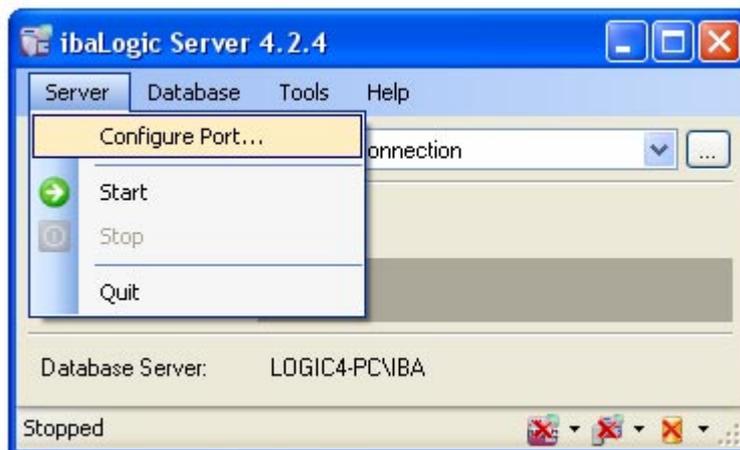
The default value of the port is set to 8086.

Prerequisite

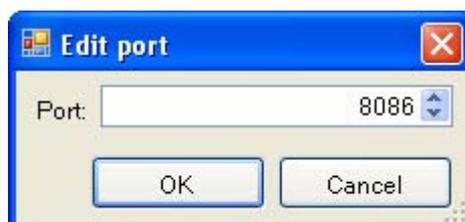
- You have stopped the ibaLogic Server.

Procedure

1. Select the menu "Server - Configure Port...".



2. Enter the port number of the client directly in the input field or set the port number using the spinner.



5.4.2 Configuring the Database Connections



Important Note

If the server can no longer connect to the local database after changing the PC name, please change the connection name from the old computer name to "localhost" under "DataSource" . (See "Configuring the Database Interface, Page 43")

You can set up multiple database connections in ibaLogic. However, only one of them is active at any given time. While installing ibaLogic, the database is created locally and the default setting pertains to this.



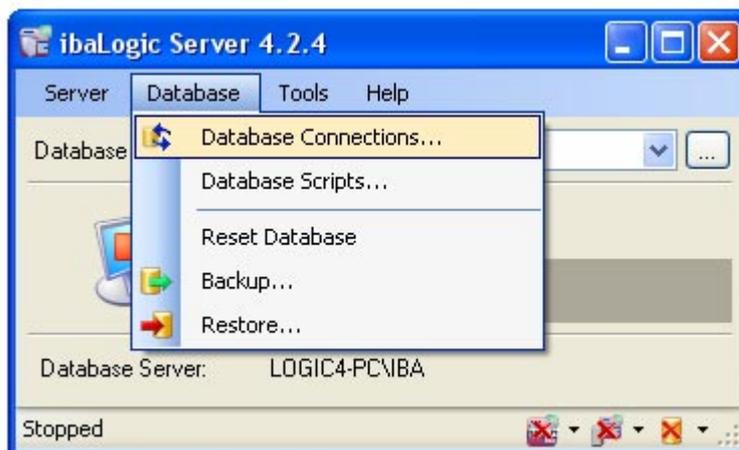
Note

You need to perform the following actions only if you want to connect to a **non-local** database and this was not already specified at the time of the installation.

5.4.2.1 Connect database

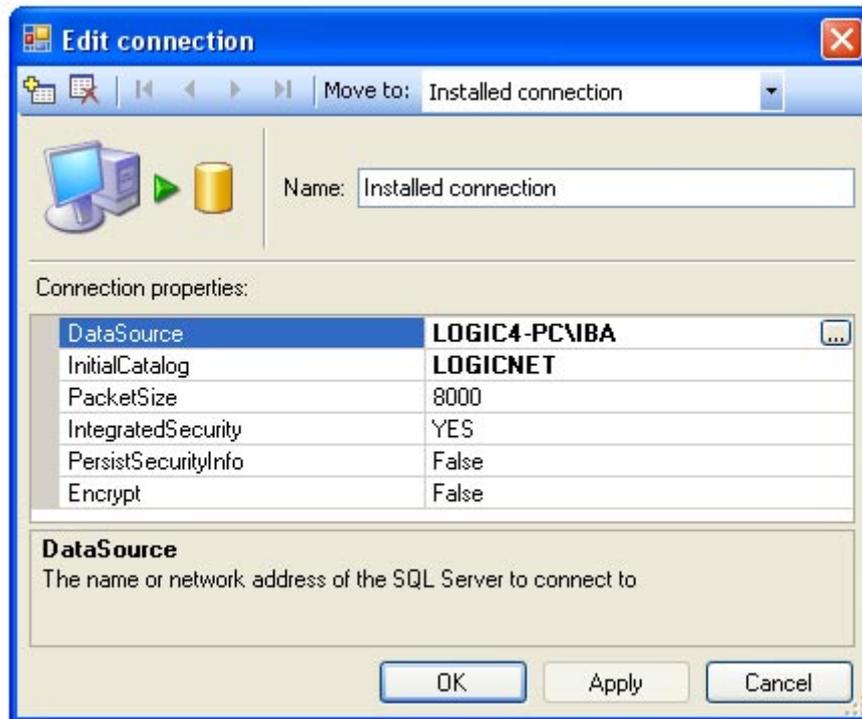
Procedure

1. Click on "Database Connections..." in the Database menu.



2. Choose an ibaLogic database connection in the "Move To" drop down box. The default setting is the "Installed connection" database connection. This is the connection to the local database.

3. Call up the configuration dialog box for database connections with the <...> browser button. The browser button becomes visible only after clicking in the text field of the "DataSource" line.



CAUTION

The following parameters

- InitialCatalog
- PacketSize
- IntegratedSecurity
- PersistSecurityInfo
- Encrypt

need to be changed only if you, e. g. wish to use a database that is already available centrally for ibaLogic also.

However, you must have basic knowledge on databases for this purpose.

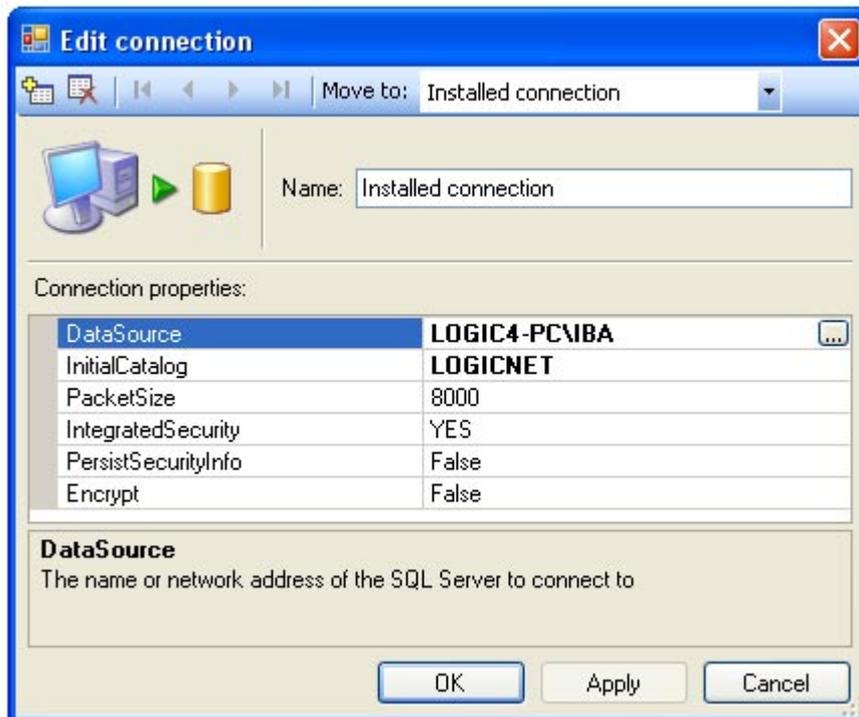
In case of doubt, please ask a database administrator to configure the settings.

5.4.2.2 Configuring the Database Interface

Enter the computer name and the instance of the ibaLogic database with which a connection needs to be established under "DataSource".

Procedure

- Enter the name of the server directly into the text field or select the database using the tree with the help of the browser button.
The browser button becomes visible only after clicking in the text field.



5.4.2.3 Select SQL server

The "Select SQL server" dialog box contains 2 tabs "Local servers" and "Network servers".



figure 9: Local server instances

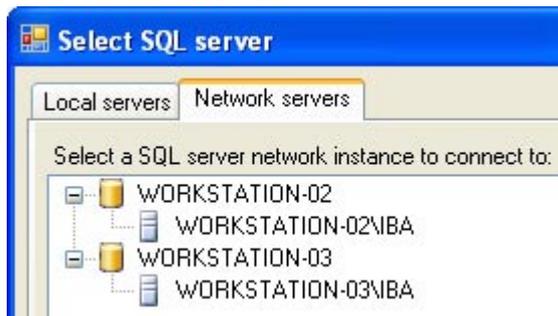


figure 10: Network server instances

All SQL database instances available on the computer are listed under "Local database instances".

After pressing on the "Network servers" tab, the entire network available is searched for SQL instances. This process may take some time. As long as the network is being searched, a message "Information is being read" appears in the text field.

Procedure

1. Click on the SQL server instance to which the ibaLogic Server should connect.
2. Finally, click on <OK>. The SQL server instance is accepted. The dialog box is closed.

Result

The SQL server instance is connected with the ibaLogic Server.

Remark

Other database connections may be installed or deleted.

The following icons are available for configuring the database connection:

Icons / Selection box	Tooltipp	Explanation
	Add	Add a new database connection
	Remove	Delete a database connection
	Start	To the first connection
	Back	To the previous connection
	Next	To the next connection
	End	To the last connection
Move to: <input type="text" value="Installed connection"/>	Choose object	Option for selecting a connection

5.4.2.4 Manage Database scripts**Important Note**

The list of installed database scripts is used to provide information for the iba support. Please do not make any modifications.

All scripts implemented in ibaLogic along with the associated information such as the version number, script name and the date of installation are displayed in tabular form when you call up the function "Database scripts". The ibaLogic Server must be stopped in order to call up the "Database scripts" dialog.

Generally, the database scripts are checked via version updates and automatically installed after a previous check.

5.4.3 Options

5.4.3.1 Activate Autostart Server

The ibaLogic Server is automatically started on Windows startup.

You can choose where the autostart options are saved:

- In the registry
- In the autostart folder of the current user
- In the autostart folder for "All users"

The default setting is that the server also starts up when the ibaLogic Server dialog is opened. If the server dialog should be open, but the server itself in STOP mode, the option "Autostart server stopped" must be enabled. In this case, the server must be started up manually so that the client connections are accepted.

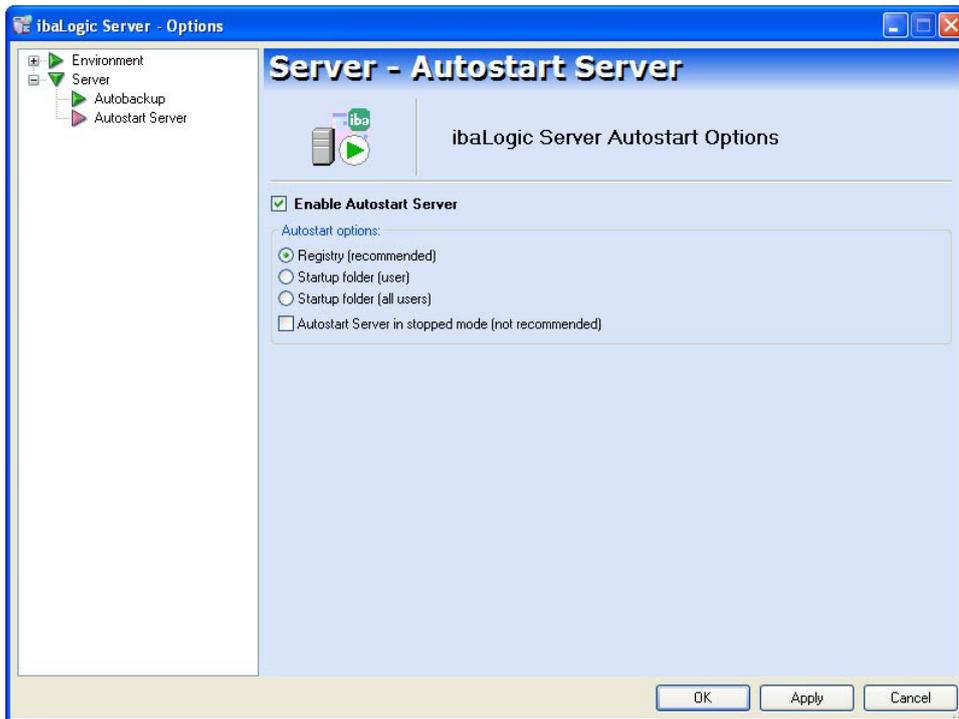
Autostart options	Explanation
Registry file (recommended)	The option saves the autostart options in the registry file.
Start-up folder (User)	The option saves the autostart options in the start-up folder of a given user.
Start-up folder (All users)	The option saves the autostart options in the start-up folder for all users.
Autostart server stopped (not recommended)	This option opens the ibaLogic Server dialog, but the server itself is not started. The server remains in the stop mode.

Procedure

1. Select the "Tools - Options" menu.



2. Choose "Server – Autostart Server" in the tree .
3. Click on the selection box "Enable Autostart Server".



4. Click on <Apply> to activate the settings.

Result

The ibaLogic Server is automatically started on Windows startup.

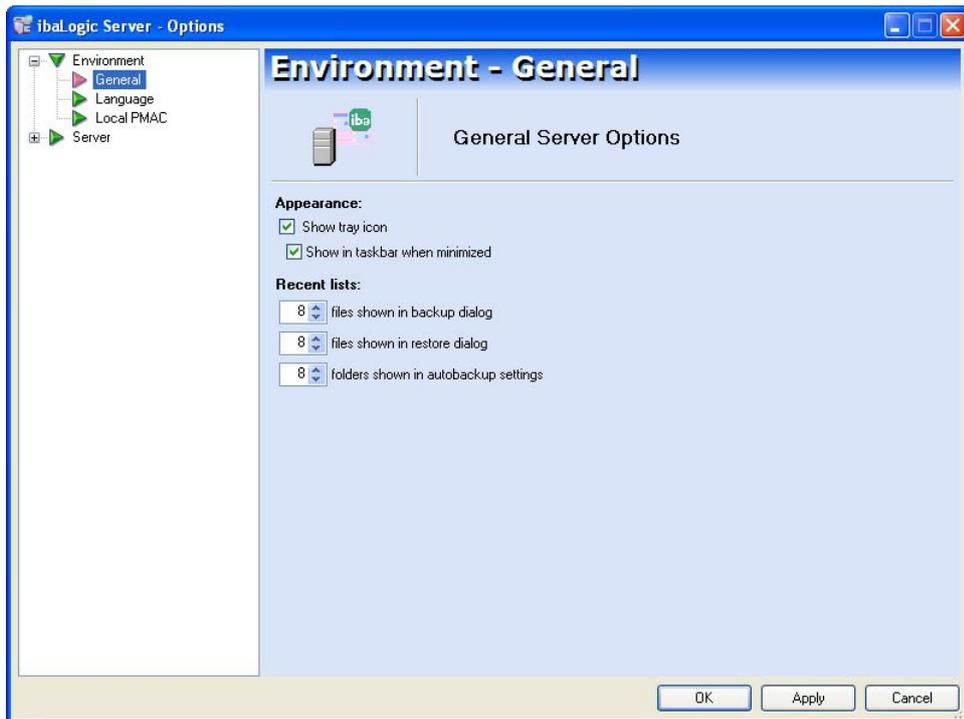
5.4.3.2 Configure General ibaLogic Server Options

The following general ibaLogic Server settings can be configured:

Server option	Explanation
Display icon in the Info section	The option, when enabled, displays an icon in the Info section when the ibaLogic Server dialog is opened.
Display in the task bar when minimized	The option, when enabled, makes the ibaLogic Server dialog appear in the task bar when minimized.
Display files in the backup dialog screen	Number of files that should be shown in the backup dialog screen. This can be chosen in the "Backup folder" selection box under "Server - Auto backup".
Display files in the restore dialog screen	Number of files that should be shown in the restore dialog screen.
Display folders in the backup settings screen	Number of folders that should be shown in the backup settings screen.

Procedure

1. Select the "Tools - Options" menu.
2. Select "Environment - General" in the tree.



3. Configure the settings as desired.
4. Click on <Apply> to activate the settings.

5.4.3.3 Settings for the Local PMAC

The local PMAC has been realized as a Windows service. Its status and start-up type (Autostart options) are configured here.

Status setting:

- Activate
- Deactivate

Status option	Description
Activate	With activate, the service, if required, is reinstalled automatically.
Deactivate	By selecting between "PMAC full version" and "PMAC demo version", it can be switched between the two run time versions. The full version requires a dongle, hardware access is not possible in the demo version, several functions are also disabled (no function in TCPIP_SendRecv, DatFileWrite, User-DLLs), even though they can be used in the project.

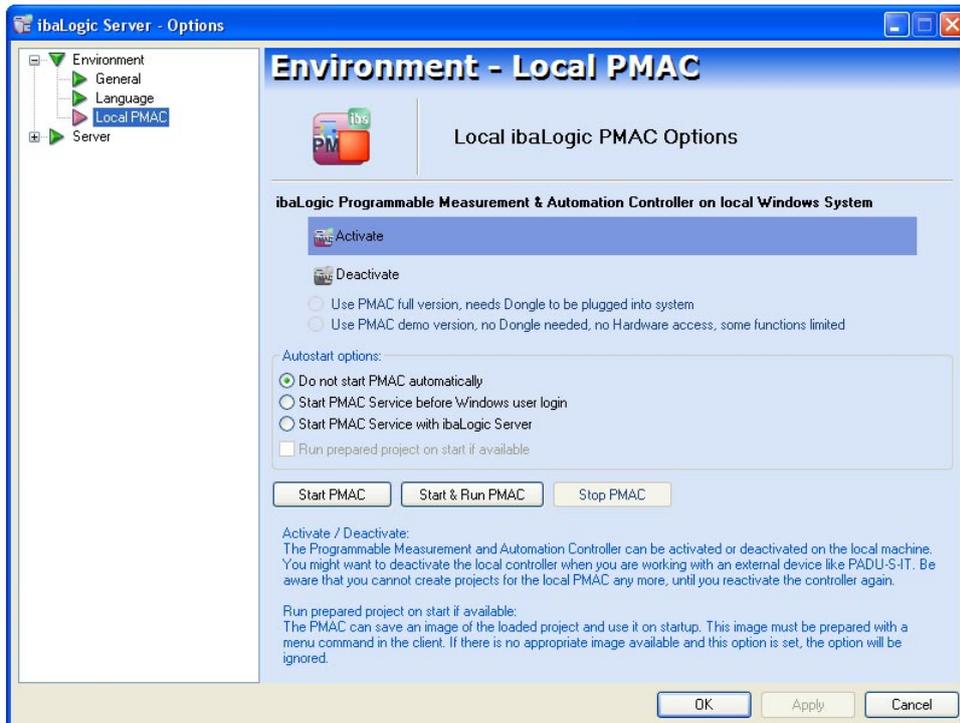
Configuring the autostart options (Start-up type):

Autostart option	Explanation
Do not start PMAC automatically	The option ensures that the PMAC service is not started up automatically.
Start PMAC service before Windows user login	This option starts the PMAC service before the Windows user login. Additional option: Run prepared project on start if available
Start PMAC service with the ibaLogic Server	This option starts the PMAC service along with the ibaLogic Server. Additional option: Run prepared project on start if available
Run prepared project on start if available	This option causes an image of a project to be loaded and used on start-up. The image must be prepared in advance via a menu command in the client. If no image is available and this option was selected, it is ignored.

Description for switching the PMAC versions (demo or full version)

Procedure

1. Stop the PMAC in the ibaLogic Server options dialog under "Environment - Local PMAC" and then disable and uninstall it.



2. Now select the requested PMAC version (full version or demo), "Activate" the service again (will be installed in doing so) and <Start PMAC> or, depending on the selected autostart option, with the next server start.
3. Select an autostart option.
If the autostart service is enabled, you can choose from the autostart options provided.
In addition, you can choose that the project should be started.
For this, it is necessary that this project has been "saved in PMAC" previously.
4. Click on <Apply> to activate the settings.

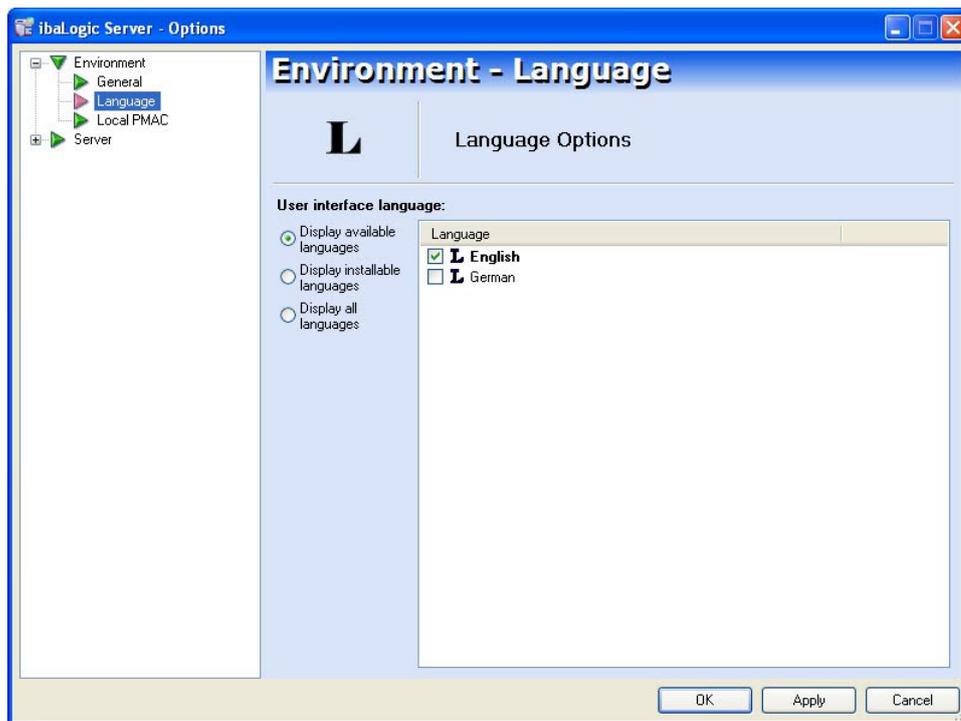
5.4.3.4 Language

Within this section, you configure the language of the server dialog.

The language of the client dialog has to be configured separately.

Procedure

1. Select the "Tools - Options" menu.
2. Select "Environment – Language" in the tree.



3. Select "Language options".
4. Choose the selection box with the desired language.
5. Click on <Apply> to activate the settings.

5.4.4 Status bar

There are 3 icons in the status bar in the ibaLogic Server dialog screen. The icons are used to activate or deactivate the autostart and backup settings.



figure 11: Functions deactivated in the status bar



figure 12: Functions activated in the status bar

Icon	Setting	Description
	Autostart (Start before login)	Activation causes the PMAC to be started automatically every time Windows starts up.
	Autostart (Registry)	Activation causes the autostart options to be saved in the registry file.
	Automatic backup	Activation causes the database to be backed up in accordance with the settings.

6 Programming Environment – ibaLogic Client

The ibaLogic Client is used to create and edit programs.

6.1 Start ibaLogic Client

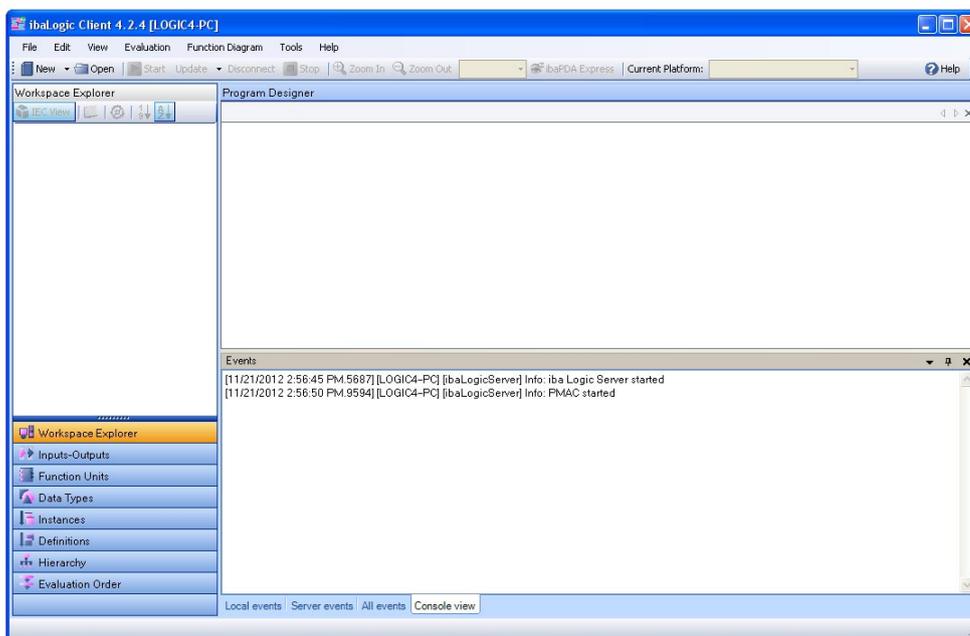
The programming environment is displayed.

Prerequisite

- You have created the start icons for ibaLogic Client and ibaLogic Server on the desktop (Standard installation).
- You have started ibaLogic Server.

Procedure

- Double click on the "ibaLogic Client" icon on the desktop.
The ibaLogic Client dialog screen opens after a brief initialization phase.



Remarks

The event window below the program window documents the program actions and collisions, if any.

6.2 User Interface of Programming Environment – Editor

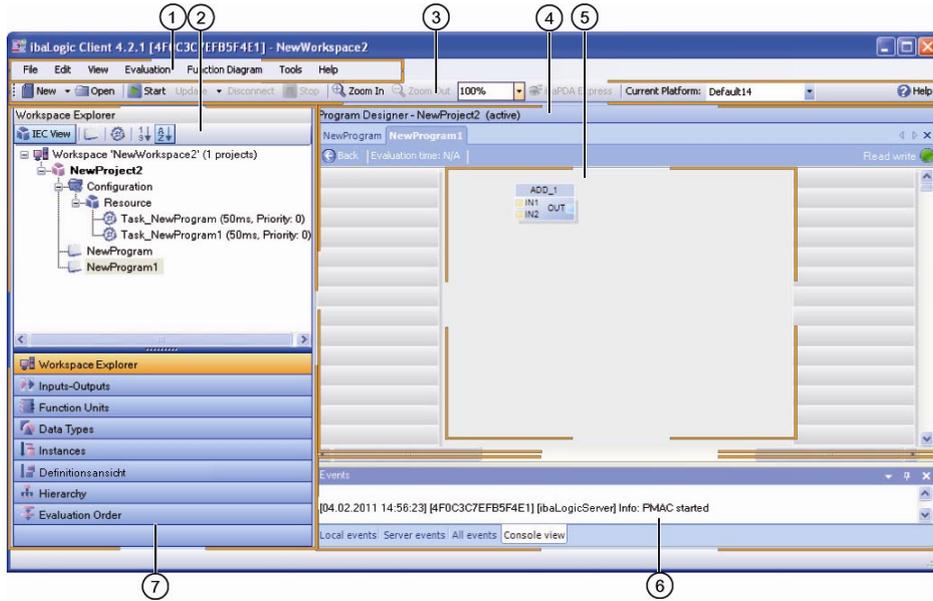


figure 13: User Interface

- | | | | |
|---|------------------|---|--------------------|
| 1 | Menu bar | 5 | Programming field |
| 2 | Navigation area | 6 | Window for events |
| 3 | Toolbar | 7 | Navigation buttons |
| 4 | Program designer | | |

6.2.1 Menu Bar

The menu bar is the central control element of the ibaLogic Client.



figure 14: Menu Bar

6.2.2 Toolbar

The toolbar is the secondary control element of the ibaLogic Client.



figure 15: Toolbar

The icons of the toolbar are also buttons at the same time.

6.2.3 Navigation Area

The navigation area contains buttons for:

- Workspace Explorer
- Inputs – Outputs
- Function Units
- Data Types
- Instances
- Definitions
- Hierarchy
- Evaluation Order

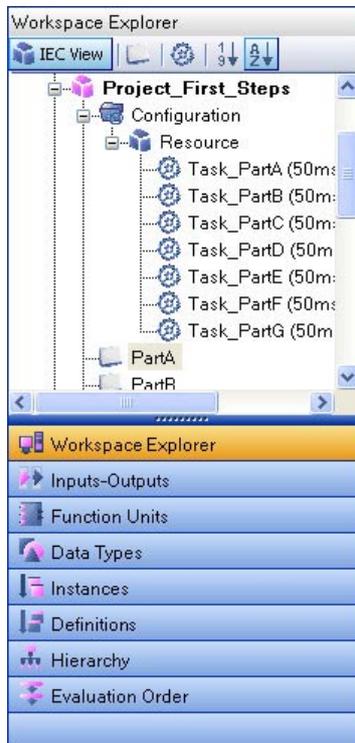


figure 16: Navigation buttons

6.2.3.1 Switch Views in Workspace Explorer

There are 3 buttons in the menu bar of the workspace explorer to switch the view of the workspace explorer. In each of the 3 views, the entries can be sorted alphabetically or according to their priority.

IEC View

This view is the default view and shows the structure of the ibaLogic working range according to standard IEC 61131-3.

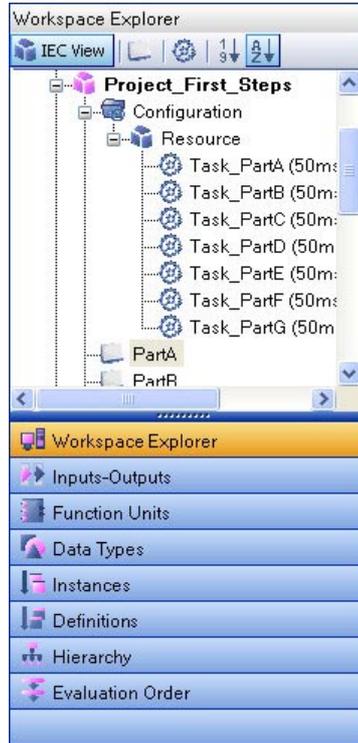


figure 17: IEC View



Note

In contrast to the IEC61131-3 standard, only one program is assigned to each task.

Prog View

Compact display of projects and programs.



figure 18: Prog View

Task View

Presentation is arranged according to tasks



figure 19: Task View

In the process, you can choose between sorting the display in alphabetical order or by priority.

Symbol	Explanation
	Sorts the entries according to names.
	Sorts the entries according to their priority.

6.2.3.2 Instances

All blocks used in the project are listed by instance names in the instances view. The definition name is also displayed, separated by a colon.

Double clicking on the instance name displays the program page in which the block is placed. The block is marked in the process.



Note on difference between Definition – Instance

The definition of a block is saved in the global library. The program always contains an instance (virtually, a copy) of the block. The instance name is automatically formed from the "Definition name/Index". However, you have the possibility to change the name afterwards.

When you modify the contents of a block, you also modify the definition and the other instances.

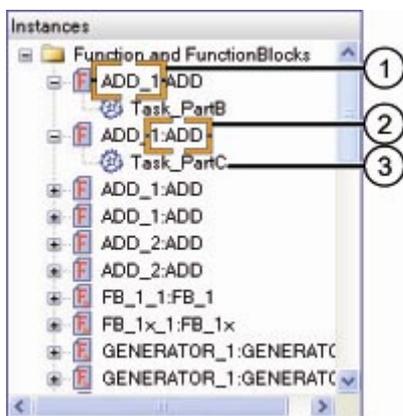


figure 20: Instances

- | | | | |
|---|-----------------|---|----------------------------|
| 1 | Instance name | 3 | Task containing this block |
| 2 | Definition type | | |

If a block is placed in a nested macro, it is displayed with a tree structure:

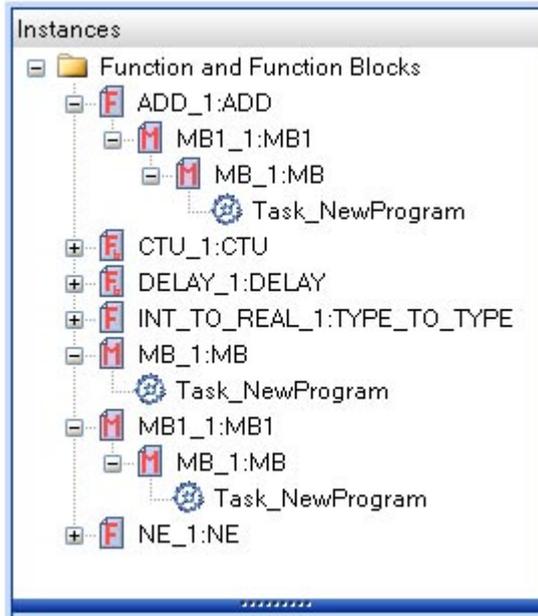


figure 21: Instance view as a tree structure

6.2.3.3 Definitions

All blocks present in the project are displayed in this view arranged in the order of their definition names. The tree structure contains instances located below the block type, and below these, macros, if any, and finally the task with the program names.

Double clicking on the instance names displays the program page in which the block is placed. The block is marked in the process.

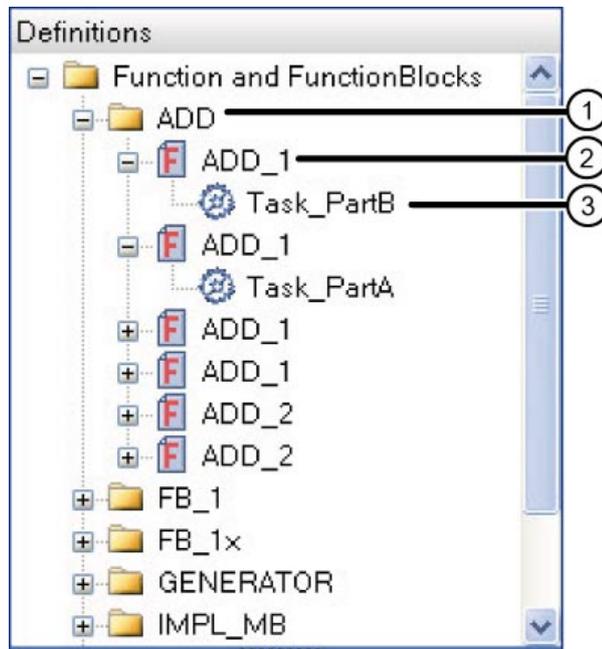


figure 22: Definitions

- 1 Definition name
- 3 Task name

2 Instance name

6.2.3.4 Hierarchy

All instances of the blocks arranged alphabetically by tasks are displayed within the hierarchy view.

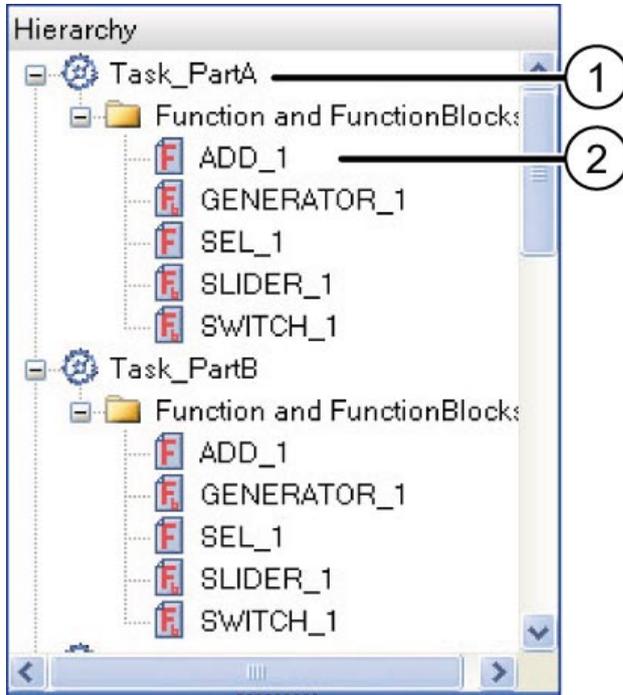


figure 23: Hierarchy view

1 Task 2 Instance name

6.2.3.5 Evaluation Order

The "Evaluation Order" view shows the sequence in which the programs and blocks are evaluated within the programs.

The blocks evaluated first are displayed at the top.

Example

2 tasks with identical interval time but different priority

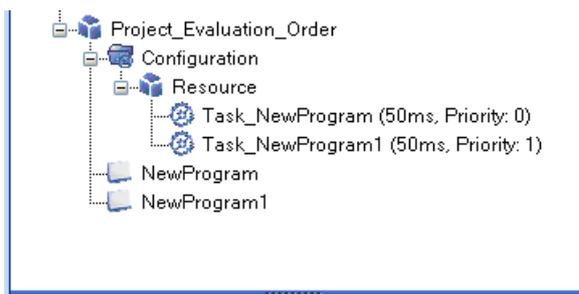


figure 24: 2 tasks with identical interval time but different priority

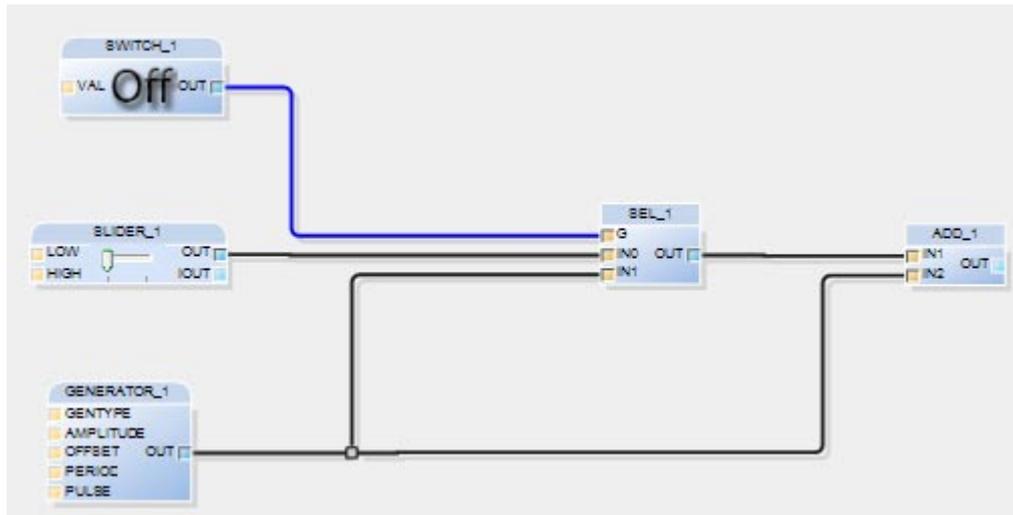


figure 25: NewProgram with the following contents

The evaluation order is displayed as follows:

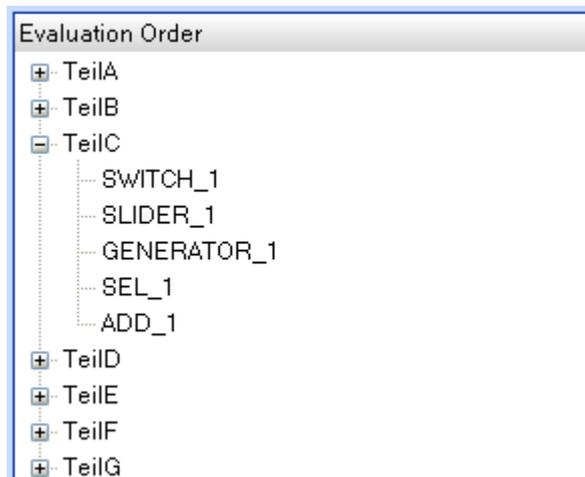


figure 26: Evaluation Order

Rules for the Evaluation Order

- The programs are executed according to the interval of the tasks assigned to them.
- Tasks that are to be started simultaneously are evaluated according to their priority, whereby 0 is the highest priority.
- Tasks are not interrupted. This means that even a higher priority task does not interrupt one with a lower priority that is running currently.
- The evaluation within a program takes place in accordance with the following rules:
 - Based on the data flow, those blocks generating the data are always evaluated first. Thereafter, those are evaluated that use the data.
 - If there are multiple independent branches in a program, the supplementary rule applicable is that evaluation takes place from the top left to the bottom right. This means that a branch located above is evaluated first.
 - In a feedback loop within a program or macro, the block placed at the uppermost left position is evaluated first.

⚠ DANGER**Danger due to modifications in the Evaluation Order!**

This has the consequence that (only in feedback loops) by shifting the blocks, the evaluation order and, thus, the results may change.

**Tip**

In order to prevent this, it is recommended to encapsulate blocks in a macro and to implement the feedback outside the macro. In this manner, the evaluation order is defined uniquely independent of the positioning.

The macro block is a block. Within the macro the blocks are evaluated in accordance with the rules given above. In other words, the output evaluated is considered as a new input value only in the next clock cycle.

**Tip**

The content of a function block is evaluated in one cycle.

This means, for example, that when you access an input connector within a For loop, you get the same value in each execution of the loop, since the connector is re-evaluated only in the next cycle. If you wish to have a loop across multiple cycles, you must obtain the feedback from the control variable external to the block.

6.2.4 Program Designer

Function blocks are placed and linked with one another in the program designer.

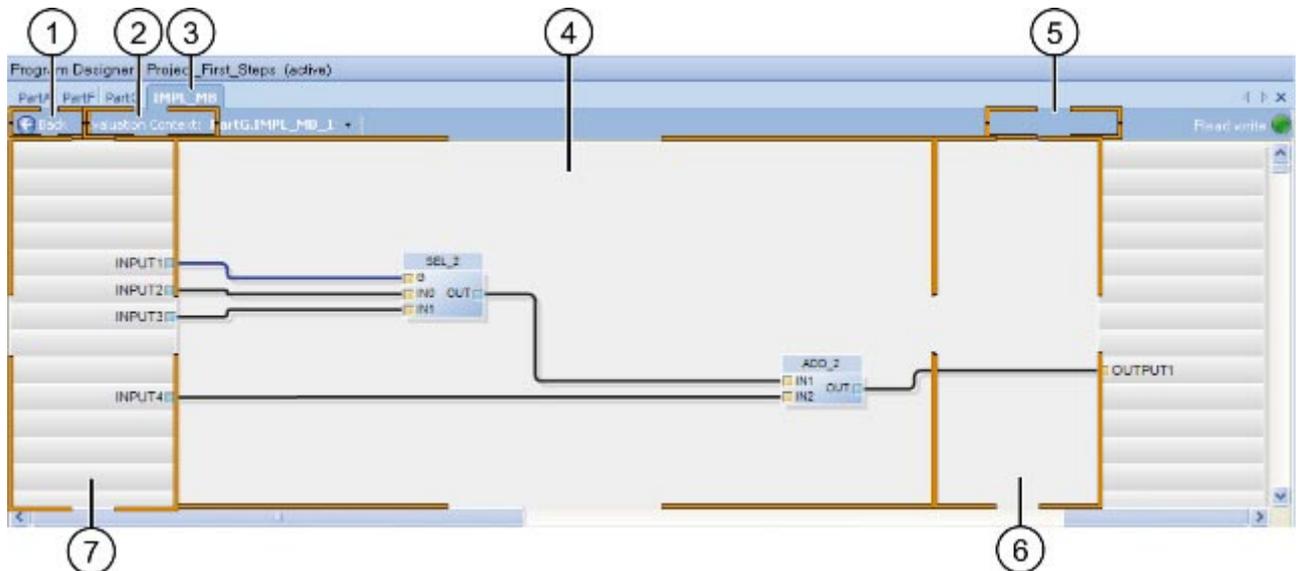


figure 27: User interface of the program designer

- | | | | |
|---|--------------------|---|--------------------------|
| 1 | <Back> button | 5 | <Release / Block> button |
| 2 | Evaluation context | 6 | Output area |
| 3 | Tab | 7 | Input area |
| 4 | Programming window | | |

6.2.5 Arrangement of the Tabs and Programming Windows

You can define the arrangement using the mouse.

The following options are available:

- Overlapping
- Horizontally or vertically

6.2.5.1 Arrange tabs

Proceed as follows to arrange the tabs:

Procedure

1. Click on the tab of the program concerned.
2. Keep the mouse button pressed, and move the tab to the new position.



figure 28: Arrangement in the form of register tabs

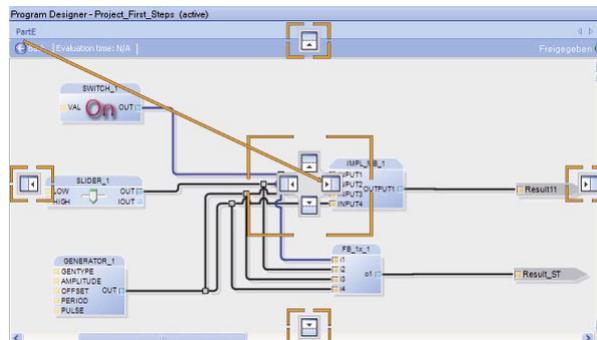
6.2.5.2 Arrange programming windows

Dockable windows are movable windows that can be shifted to any position on the screen and can be coupled to the docking marks.

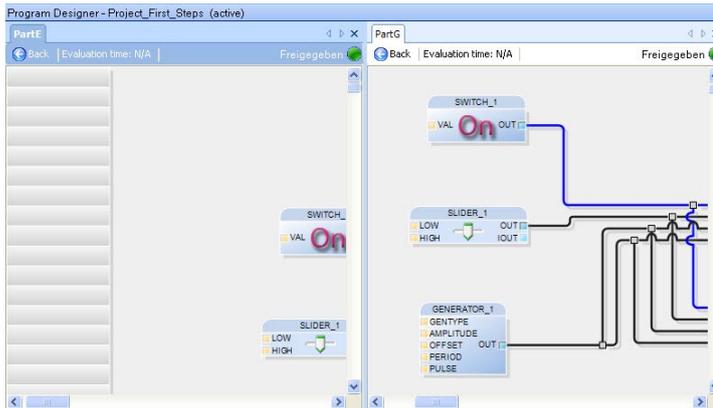
Proceed as follows to arrange the windows:

Procedure

1. Move the mouse pointer to the tab of the window that you wish to shift.
2. Press the left mouse button and keep it pressed.
3. Move the window to the docking mark at which you wish to position the tab.



4. Release the mouse button. The window is now anchored at this position.



Note

Program windows cannot be positioned as desired.

The programs and macros that are opened are displayed graphically in the programming field. Since macro blocks are displayed exactly in the same way as programs, they are not described in the following sections.

Toolbar

You can use the following functions in the toolbar:

- Back ()
- Evaluation context (only with macros)
- Evaluation time (only in the online mode)

Back

You can use this button to navigate in the program window. The last selected program / macro is displayed.

Evaluation context (only with macros)

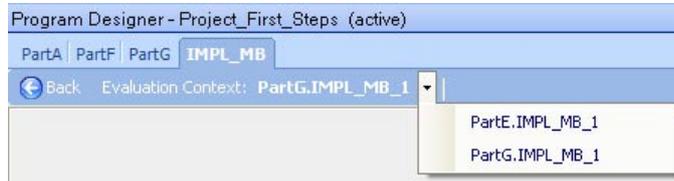
It is used to display the program or macro level in which the current plan is located. Since the instances of a macro block can occur several times in one or in various programs with different call parameters, you can see the program and instance in which the macro currently open is located.

If one macro is located in another macro (nesting), the entire hierarchy branch is displayed separated by dots:

Display: Program_name.Macro_1.Macro_2...

You can select the context within an open macro, i. e. you can switch from one instance to another. The values seen in the online view are always related to the instance selected.

1. To do this, click on the arrow button on the right side beside the macro instance name.



2. You can select the calling level from within the macro when you click the instance name in the evaluation context directly.



The area of the program field window that contains this macro instance is displayed and the macro block is marked.

Evaluation time (online mode only)

It is displayed in each program as a percentage value with respect to the task interval and as an absolute value in ms (smoothed mean value).

6.2.5.3 Navigating in the Program Designer

You have several options to navigate in a program:

- Zoom
- Program overview
- Hot keys

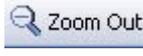
Zoom

You have the option to enlarge (zoom in) or reduce (zoom out) the size of the programming field in the program designer.

The following possibilities for zooming out or zooming in are available:

- Use of the buttons <Zoom in> / <Zoom out>
- Use of the picklist
- Use of the hot keys

Buttons

- Press the button.  or  in the ibaLogic Client toolbar, to zoom in or zoom out by 20 % at each click the view in the programming field of the program.

Pick list

- Select  between the 5 preset zoom-values in the pick list
- or
- Enter a zoom factor using the keyboard from 25 to 200 percent in the drop down box (pick-list).

Key combination

- Press <Ctrl> and roll simultaneously the scroll wheel of the mouse.

The minimum zoom factor depends on the screen resolution.

Program overview

Since one program page generally exceeds one screen page and only a partial window of the program can be seen depending on the zoom factor, the program overview serves as an orientation aid.

Open and use program overview

Procedure

1. Select "View - Program Overview" in the main menu.

A miniature view of the "Program Overview" window is opened in the foreground of the current program window.

The visible section is displayed as a transparent rectangle.

2. Move the rectangle by using the mouse until you see the desired program area in the program designer.

Annotation

The overview window can be placed arbitrarily.

It can also be docked outside the programming field or at its margin.

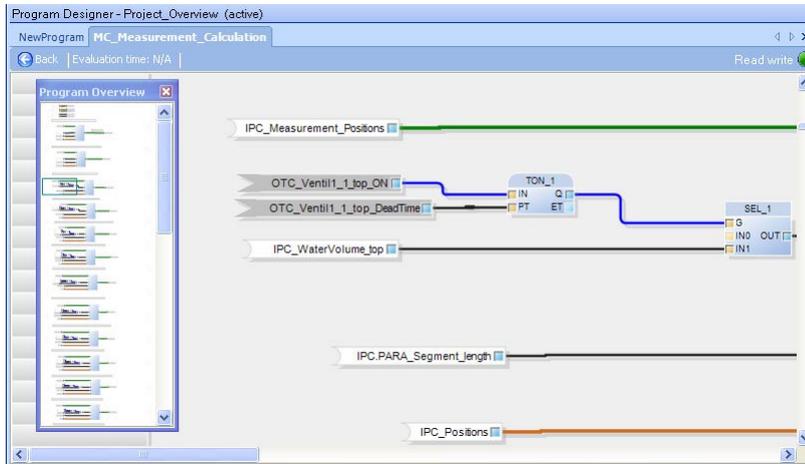


figure 29: Programming field with displayed program overview window

Navigate in the programming field

The navigation in the program window is generally done with the mouse.

In the following, the mouse functions are explained in summary:

- Scroll wheel: Scrolling of the visible area to the top/bottom
- Scroll wheel + <Shift>: Scrolling of the visible area to the left/right
- Scroll wheel + <Ctrl>: Zoom out/zoom in

Enlarge page to the bottom

The program page has a default width of 2500 pixel and a default height of 10000 pixel. The visible cutaway depends on the screen resolution.

If the page size is not sufficient, enlarge the page to the bottom.

Processing

1. Open the context menu by clicking with the right mouse button on a free area in the programming field.
2. Select "Extend page (vertical)" in the context menu.

Result

From the current mouse cursor position, a range of 800 pixel lines is added.

Annotation



Important note

Please ensure that there are no blocks in the horizontal line of the mouse pointer. Connecting lines that cross this intended horizontal line are extended.



Note

You cannot remove an empty page within the task using any function. An empty area at the lower end of the task is removed automatically when loading a project.



Tip

An empty space within the tasks can be achieved by reducing the zoom-level and shifting the objects jointly.

6.2.6 Synchronize Access (<Read write>/<Read only> buttons)

During the multi-client operation the access to windows and dialogs can be synchronized by the buttons <Read write>/<Read only>.



figure 30: <Read write> button

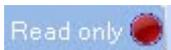


figure 31: <Read only> button

6.2.7 Events Window

The "Events" window below the "Program Designer" window documents the program actions and collisions, if any.

The events window is divided into 4 views with the help of tabs.

You can choose from the following views:

- Local events
- Server events
- All events
- Console view

Event icons are used in the "Local Events", "Server Events" and "All Events" views.

Event icon	Event	Level	Description
	Info	Info	Displays information
	Warning	Warning	Displays a warning
	Error	Exception	Displays an error

6.2.7.1 Local Events

You can use the "Local Events" view to display events of the client in a formatted manner.

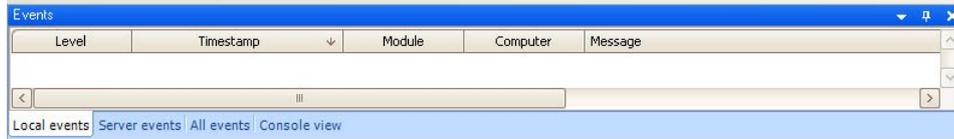


figure 32: Events window – "Local Events"

6.2.7.2 Server Events

The "Server Events" view is used to display events pertaining to the server.

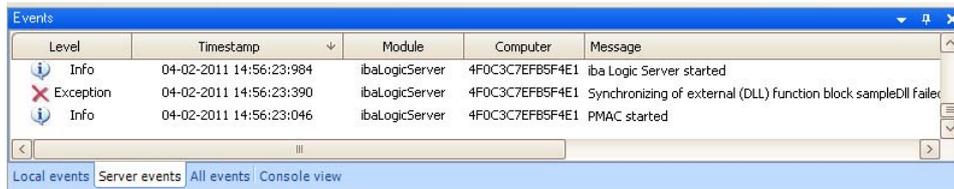


figure 33: Events window – "Server Events"

6.2.7.3 All Events

You can use the "All Events" view to display events of the client and the server in a formatted manner.

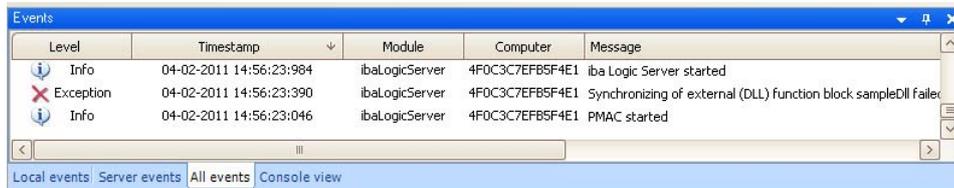


figure 34: Events window – "All Events"

6.2.7.4 Console View

The "Console View" is used to display all events as simple text sorted chronologically according to their occurrence with the corresponding explanation.

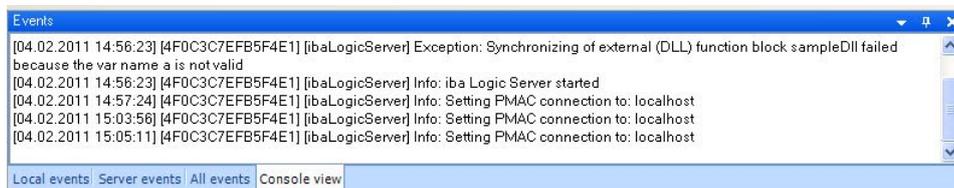


figure 35: Events window – "Console View"

6.3 Workspace

You can use a workspace to save programs and projects in a sorted manner.

6.3.1 Create Workspace

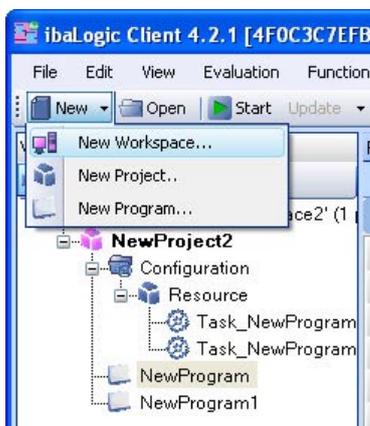
Prerequisite

You have not opened any workspace.

You can close any workspaces that are open via "Close workspace - file".

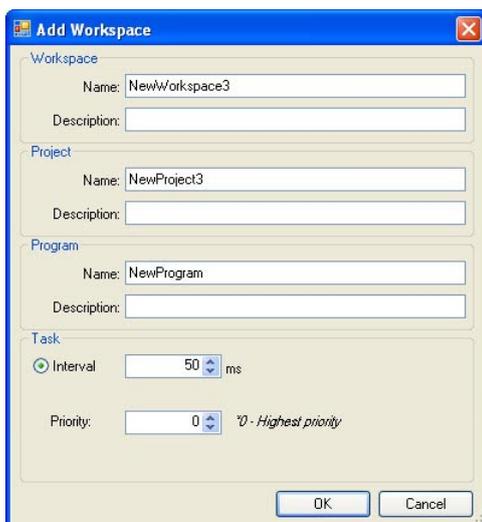
Procedure

1. Click on the arrow of the <New> icon in the toolbar.
2. Select the "New Workspace" menu entry from the list.



The "Add Workspace" dialog box is displayed.

3. Assign a name and a description for the program.
ibaLogic creates a new project and a task automatically. Assign meaningful names and descriptions that also conform to the IEC standard.
If required, define also the interval time and its priority for the task.



Remark

You can modify all settings later on.

You can modify the settings via the respective context menu for "Properties".

A name that has already been assigned is displayed with an "X" at the end of the input field.



You can fill the fields for the description with any comments as you please. These comments are displayed above the object as a tooltip.

**Notes**

The names must comply with the IEC standard. For further information refer to "Naming conventions, Page 277".

**Tip**

You can set the default value for the names under "Tools – Options – Editors – Workspaces".

6.3.2 Open workspace

Procedure

1. Press the <Open> button. The "Open Workspace" window is displayed.
2. Select the desired workspace and click <OK>.

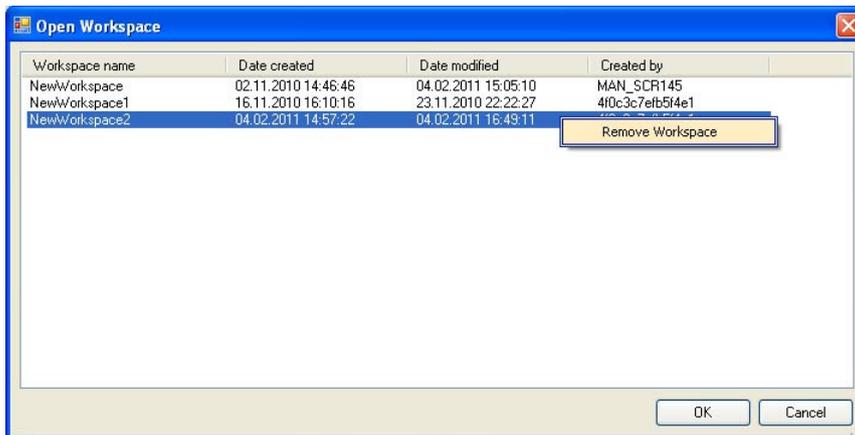
6.3.3 Close Opened Workspace

Select the workspace in the "File – Close Workspace" menu.

6.3.4 Remove Workspace from the Database

Procedure

1. Click on the <Open> button. The "Open Workspace" window is displayed.
2. Mark the workspace that needs to be removed.
3. Open the "Remove Workspace" context menu by clicking on the right mouse button.



4. Click on "Remove Workspace". The "Remove Workspace" dialog box is displayed.
5. Confirm with <Yes>.
6. Confirm the procedure with <OK>. The dialog box is closed.

6.4 Workspace Projects

ibaLogic automatically creates a project while creating a new workspace. You can create more than one project within one workspace.

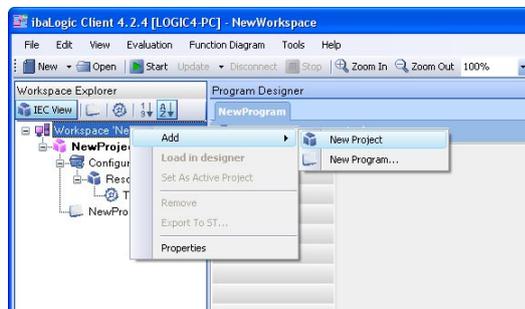
Prerequisite

- You have started the ibaLogic Server and the ibaLogic Client.
- You have created at least one workspace.

6.4.1 Create Project

Procedure

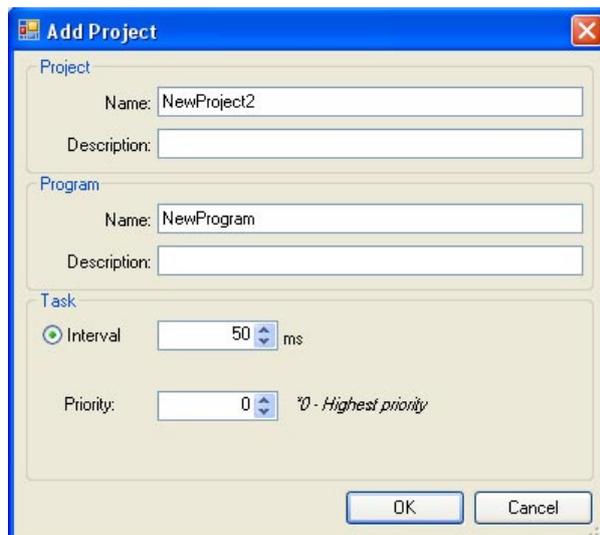
1. Click with the right mouse button on the workspace name to which the project needs to be added in the Workspace Explorer.
2. Select "Add - New Project" in the context menu.



The "Add Project" dialog box is displayed.

The dialog box that opens is the project-specific section from the workspace dialog.

Assign a name to the project and the program. Assign meaningful names that conform to the IEC standard. If required, set up the interval time and priority for the task.



6.4.2 Set Project as Active

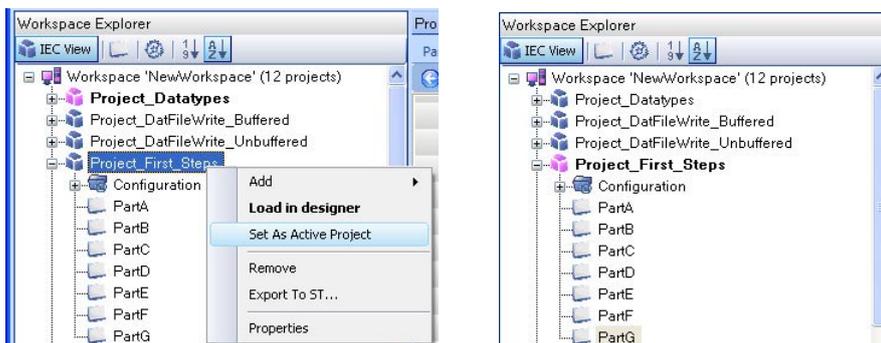


Note

Only one project of many can be active within a workspace.

Procedure

1. Open the context menu of the project to be activated by clicking on the right mouse button.
2. Select "Set As Active Project".



Result

The name of the active project is displayed in the Workspace Explorer in **"Bold"** and the associated icon changes its color from blue to pink.



Notes

If any project is in the online mode (programming field background is pink), no other project can be set as active.

By setting a project as active, the programs of the project are not loaded automatically in the design area, but need to be selected manually as required.

The buttons in the toolbar and in the Workspace Explorer are only valid for the active project. These are:

- <Start>
- <Stop>
- <Disconnect>
- <Current target system>
- <I/O configurator>

6.4.3 Load Project in the Program Designer

Regardless of whether a project is active or not, you can load programs in the design area.

Procedure

1. Mark the project that you would like to load in the designer.
2. Select "Load in Designer" in the context menu.

Result

The programs loaded are displayed as tabs at the upper border of the design area.

6.4.4 Edit Project Properties

You can modify the name and description field in the project properties. You can enter supplementary information about the project in the "Description" text field.

Procedure

1. Click with the right mouse button on the project whose project properties you would like to edit.
2. Select "Properties" from the context menu.
The "Edit" window is displayed.
3. Edit the properties of the project.
4. Click on <OK>.

6.4.5 Remove Project

The project selected is removed from the workspace and from the database at the same time.

Procedure

1. Click with the right mouse button on the project that you would like to remove from the workspace.
2. Select "Remove" in the context menu. The "Remove Project" dialog box is displayed.
3. If you are sure that you would like to remove this project from the workspace, click on <OK>.
4. If the project has been deleted, the "Project removed" dialog is displayed for confirmation.
5. Finally, click on <OK>.

6.5 Tasks/Programs

ibaLogic creates a program automatically while creating a new project. You can add other programs to a given project.



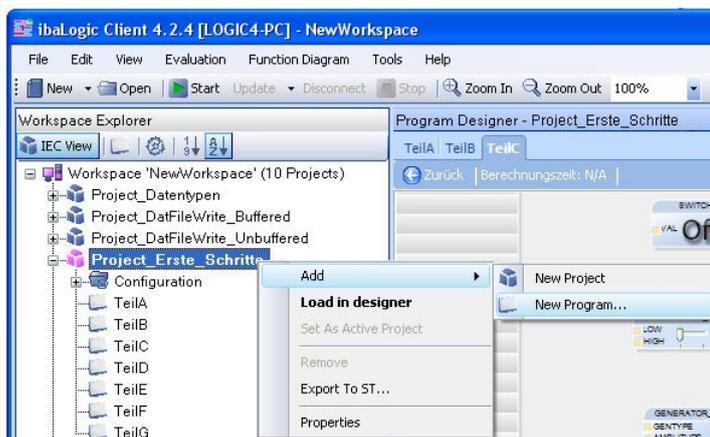
Notes

Please keep in mind that you cannot delete the last task. One task must always be present for each project.

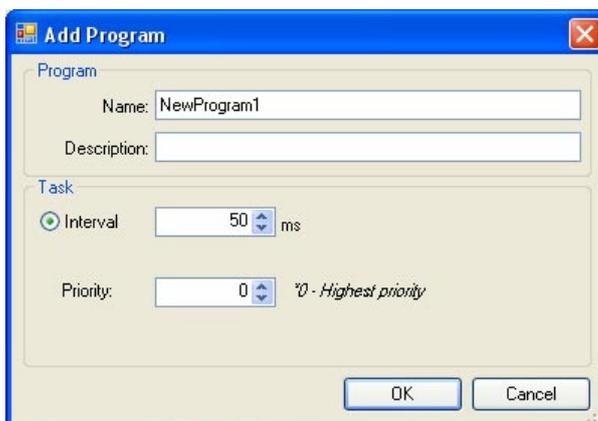
6.5.1 Create Tasks / Programs

Procedure

1. Click with the right mouse button on the project in which you would like to create a new program.
2. Select the "Add - New Program" from the context menu.



The dialog "Add Program" is displayed.



3. Assign a name and a description for the program. An associated task is created at the same time. Assign meaningful names that comply with the IEC standard. Further information see definition in "Naming conventions, Page 277".
4. Click on <OK> to add the new program.

Exactly one task is assigned to each program. You can choose from the following task parameters:

- Interval time
- Priority

Interval

The program belonging to the task is restarted exactly after the specified interval time.

If, under extreme circumstances, the evaluation time (or the evaluation of all programs) is greater than the specified interval time, the system is overloaded. How to proceed is explained in "Time behavior, Page 230".

The default value is 50 ms. The smallest interval possible is 1 ms, but the time interval cannot be less than the timebase set in the I/O configurator.



Notes

You can change the default values for the program name and the time interval under "Tools – Options – Editors – Workspace".

Priority

Each task is assigned a priority. "0" means highest priority.

Please note that no task can be interrupted by another having a higher priority. The priority is relevant only for the order of evaluation. For more information, please see "Time behavior, Page 230".



Notes

You can display the tasks either in alphabetic order or in the order of their priorities in the workspace. You can use the icons at the upper border of the navigation section for this purpose



6.5.2 Open Tasks/Program

Double click on the program name in the Workspace Explorer.

6.5.3 Change Task / Program Properties

Change the properties of existing tasks / programs.

Procedure

1. Click with the right mouse button on the corresponding task or program.
2. Select "Properties" in the context menu. The "Edit" window is displayed.
3. Change the properties.
4. Finally, click on <OK>.

6.5.4 Remove Task / Program

Remove a program / task from a workspace.

Procedure

1. Click with the right mouse button on the corresponding task or program.
2. Select "Remove" in the context menu. The "Confirm" dialog box is displayed.
3. If you wish to remove the program, click on >Yes>.



Notes

Please keep in mind that you cannot delete the last task. One task must always be present for each project.

6.5.5 Import / Export Programs

To exchange complete programs between projects of different data bases exist export/import functions.

Export procedure

1. Click with the right mouse button on the correspondent program.
2. Select "Export to ST" in the context menu. The "Export" dialog box is displayed.



3. Call up the file browser, select a directory, enter the filename and click on "Save".
4. Enable the option "With additional graphical information..." and click on OK.

Import procedure

1. Switch off the calculation because the import is only available in the offline mode.
2. Select "File-Import-Structured Text" under main menu.
The "Structured text import" dialog is displayed.
3. Call up the file browser and select directory and file (with file extension *.il4) and press "Open".
4. Select the option "Import definition of Function blocks as new" and click on OK.

During the import, the names of the function blocks and data types to be imported are compared to already existing names. If names are already assigned, the definitions are overwritten (TRUE) in relation to the option: "Import definition of function blocks as new" or a new definition is created with name+index (FALSE).

The imported program is newly created together with the task. If the program name is already assigned, they are created with name+index.

During the import of a project, all programs and tasks in the project are newly assigned, probably with "name+index".

**Note**

For export/import description for **blocks** refer to "Exporting Blocks, Page 93" or "Importing Blocks, Page 94".

6.6 Configure Inputs and Outputs

Inputs and outputs are routed signals to peripheral devices.

You work with "virtual" signals in the programming environment. You must assign these using an allocation process to hardware signals that are actually present.

You can do this assignment from the viewpoint of the program or that of the hardware, for each signal separately or for groups of signals.



Important note

For more information, please refer to "IO Configuration, Page 175".

The inputs and outputs of the hardware are arranged in a tree structure within the navigation section of the I/O configurator.

The hierarchy levels are:

Direction → group → signal subdivision → signals

- Direction:
"Inputs" or "Outputs"
- Group:
group name either created manually or taken over from the module name of the signal assignment.
- Signal subdivision:
The signals are divided in analog and digital signals.
- Signals:
 - signal name either created manually or taken over from the name of the signal assignment.
 - behind that the data type in brackets,
 - then the hardware signal name to the right of the arrow (->)

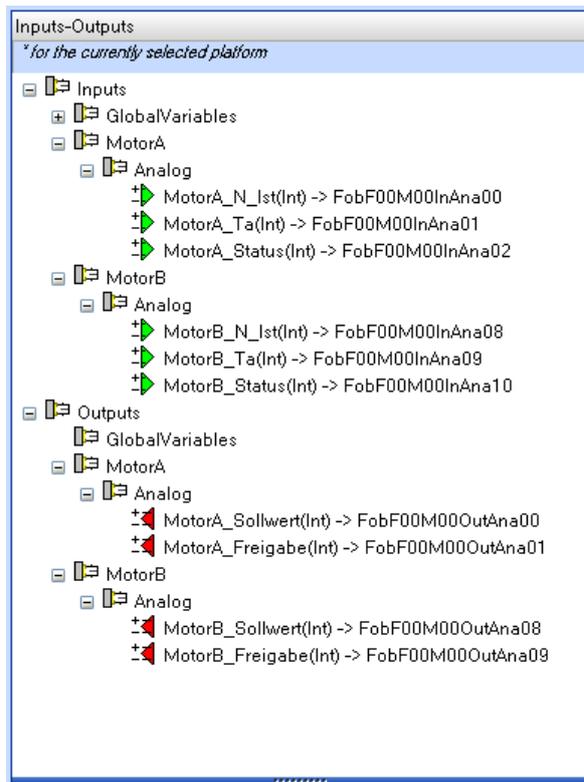


figure 36: Inputs - Outputs

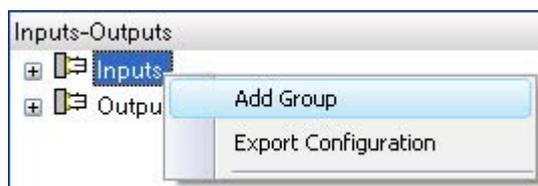
6.6.1 Create Signals

Signals are assigned to groups. This facilitates meaningful structuring.

6.6.1.1 Define Group

Procedure

1. Click with the right mouse button on "Inputs - Outputs" and select the menu item "Add group".



2. You determine the name for the new group in the dialog "Setup group name".

Result

The new group is created under "Inputs" as well as "Outputs".

Groups which are not necessary can be deleted by means of the context menu and/or the DELETE key.

Example

Groups: Motor A, Motor B

Input signals: Actual speed, motor temperature

Output signals: Set-point value, parameter

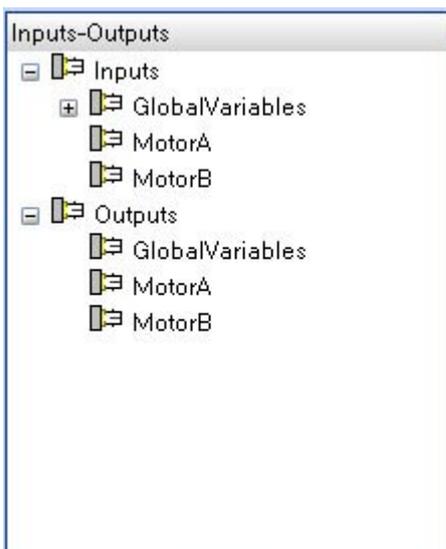


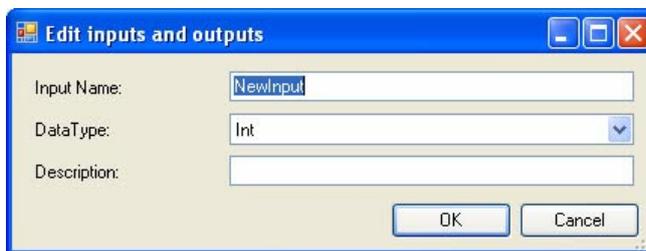
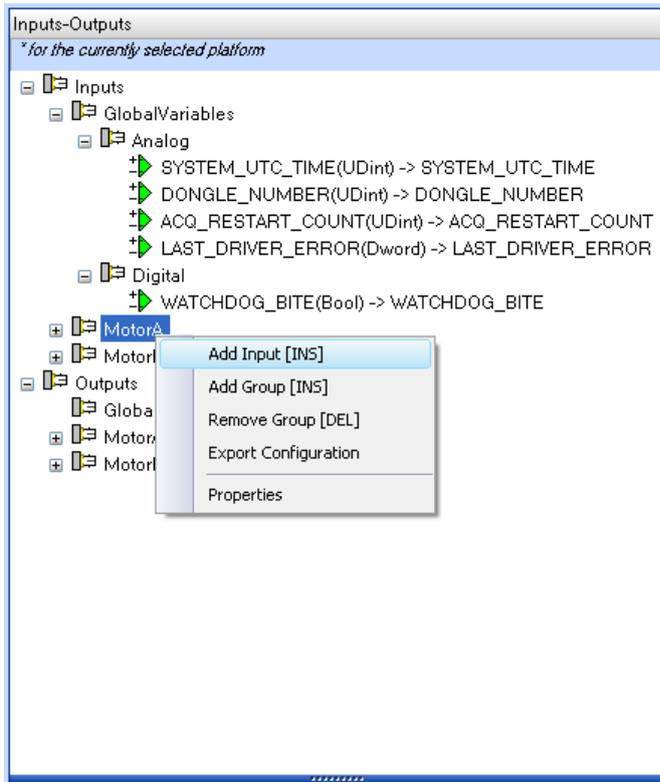
figure 37: View "Inputs – Outputs"

6.6.2 Define Signals

Procedure

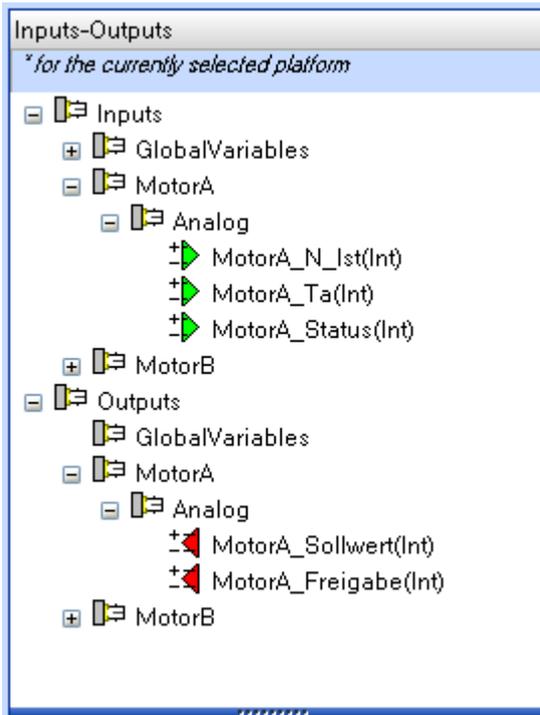
1. Click with the right mouse button on an input or output group.
2. Select "Add input" or "Add output" in the context menu.
The "Edit inputs and outputs" dialog will be opened.

3. Assign a signal name, data type and, a description, if any. Assign meaningful names that conform to the IEC standard.



Please note that for the data type, this is provided by the peripheral device. If the analog value from an ibaPADU-8 is used, for example, it is always an integer value.

In our example, the following scenario would emerge:



The signals have not yet been assigned to any hardware.

6.6.3 Edit Existing Signals

Procedure

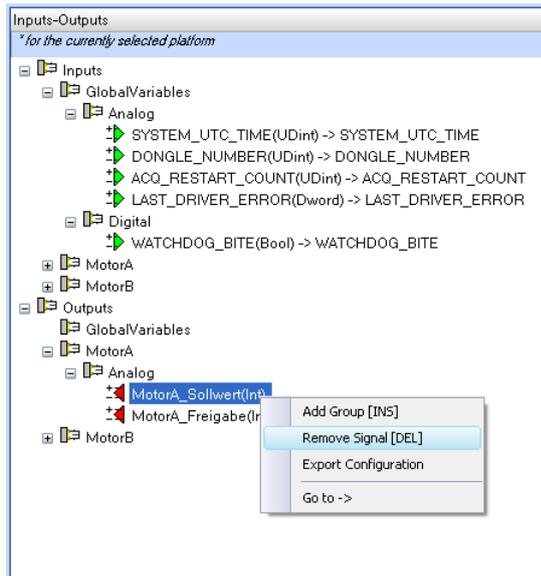
1. Double click on a signal that has been defined.
The editing dialog box opens.
2. Assign a signal name, data type and, a description, if any. Assign meaningful names that conform to the IEC standard.
3. Click on <OK> to accept the modifications.

6.6.4 Remove Signals

Procedure

1. Click with the right mouse button on a signal that has been defined.
The context menu opens.

2. Select "Remove Signal" in the context menu.



Note

You can edit and remove inputs and/or outputs only if you do not yet use them in the program, i. e. they are not yet visible in the input and/or output bar.



Note

The assignment of the signals defined here to the hardware available is described in the "Signal assignment, Page 182" section.



Important Note

The interface cards of third party manufacturers sometimes do not provide individual signals with the elementary data types, REAL and INT. A data structure as an input / output is generated here for the Profibus master card SST, and it depends on the configuration of the slave (GSD file). You must also define this structure in ibaLogic to be able to use the signals included there.

An example of the connection of the Profibus master card is documented and included in the CD supplied.

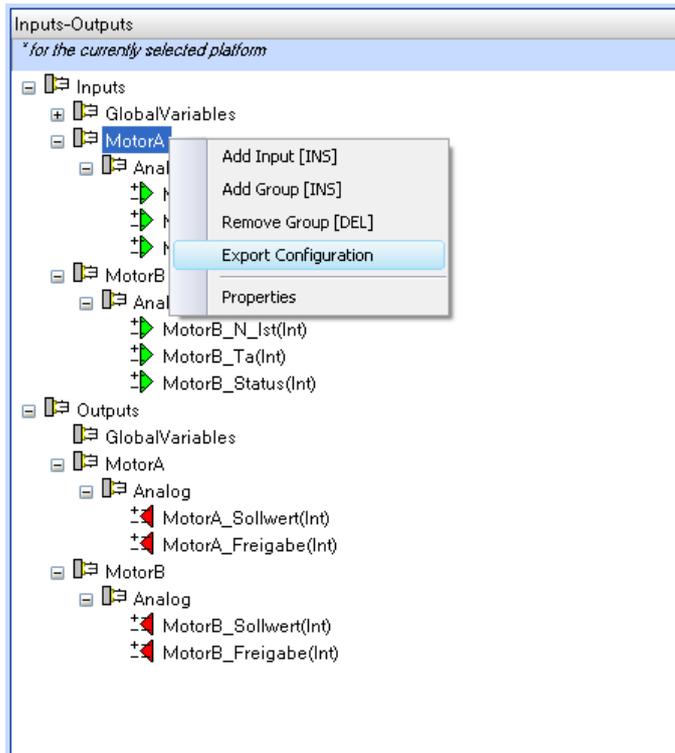
6.6.5 Export / Import Signals

The virtual signal names are often already provided in external documents. Use the export and import functions in order to use them.

Exporting the entire I/O configuration

Procedure

1. Open the context menu of a group or of a signal and select the menu option "Exporting configuration".

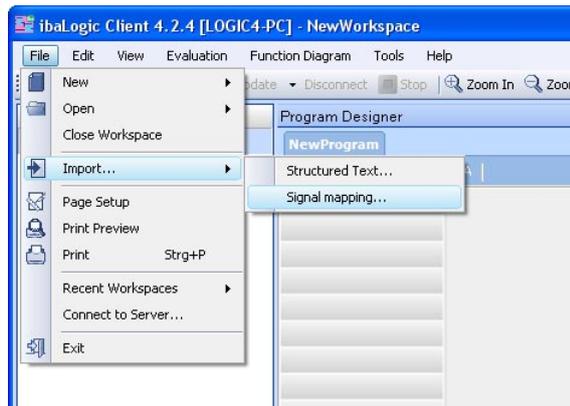


2. Open the export, for example, in a spreadsheet program.
3. Enter the virtual group designation and signal name in the group and icon columns or modify the existing names.

```
Adresse; Typ; InOut; Gruppe; Symbol; Kommentar
FobD00M00InAna00; INT; Input; FobD00M00; FobD00M00InAna00;
FobD00M00InAna01; INT; Input; FobD00M00; FobD00M00InAna01;
FobD00M00InAna02; INT; Input; FobD00M00; FobD00M00InAna02;
FobD00M00InAna03; INT; Input; FobD00M00; FobD00M00InAna03;
FobD00M00InAna04; INT; Input; FobD00M00; FobD00M00InAna04;
FobD00M00InAna05; INT; Input; FobD00M00; FobD00M00InAna05;
FobD00M00InAna06; INT; Input; FobD00M00; FobD00M00InAna06;
FobD00M00InAna07; INT; Input; FobD00M00; FobD00M00InAna07;
FobD00M00InAna08; INT; Input; FobD00M00; FobD00M00InAna08;
```

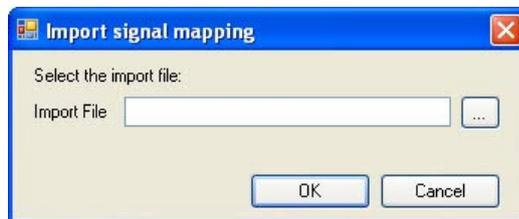
	A	B	C	D	E	F
1	Adresse	Typ	InOut	Gruppe	Symbol	Kommentar
2	FobF00M00InAna00	INT	Input	FobF00M00	FobF00M00InAna00	
3	FobF00M00InAna01	INT	Input	FobF00M00	FobF00M00InAna01	
4	FobF00M00InAna02	INT	Input	FobF00M00	FobF00M00InAna02	

4. Import the signal assignment by using the menu item "File-Import-Signal assignment".



The "Importing signal assignment" is displayed.

5. Specify the file name with the target path.



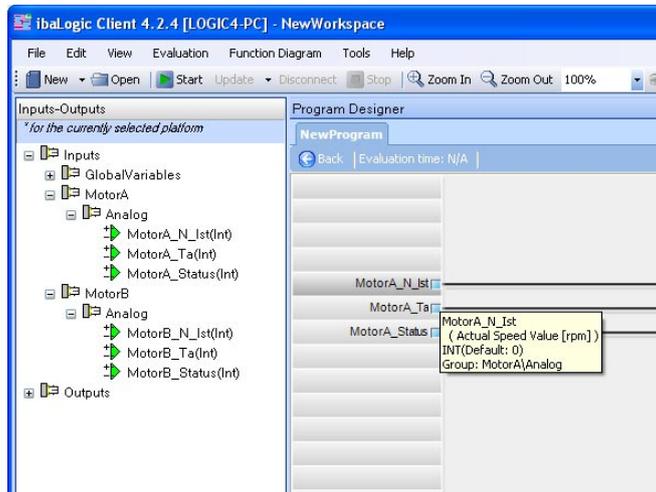
6. Click on <OK> to start the import.

6.6.6 Using Signals in the Program

To be able to use signals in a project program these must be dragged to the input or output sidebar.

Processing

- Drag the desired individual signal or the entire group to the sidebar of the input or output.

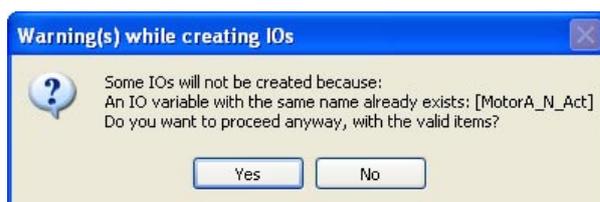


Example

The signal "MotorA_N_Ist" has been dragged to the input sidebar.

When you point with the mouse on the connector symbol of the signal, a tooltip is displayed with information pertaining to this signal.

If you drag an entire group to the sidebar, all individual signals are placed there. If certain signals of the group are already placed in the sidebar, the following warning message is output.



Note

For further information please see "IO Configuration, Page 175".

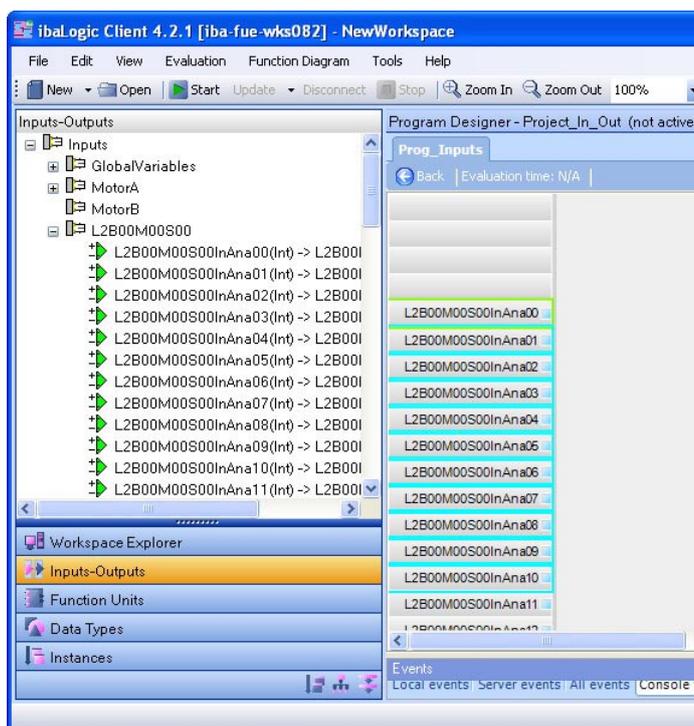
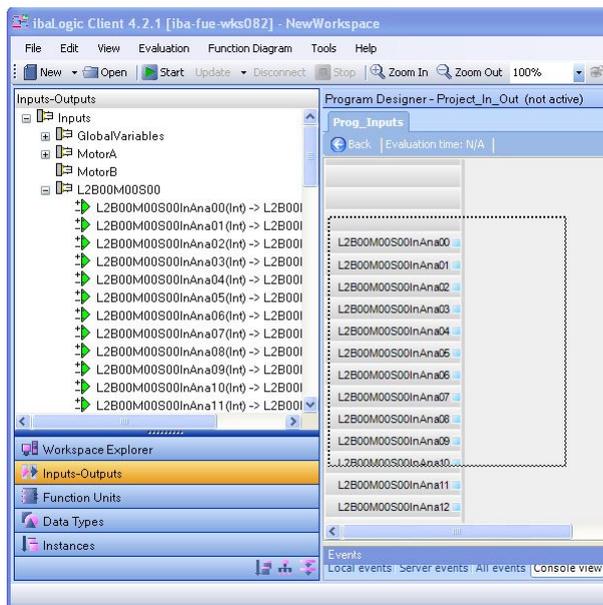
6.6.7 Remove Signals in the Program

Procedure

1. Mark the signal in the input or output bar.
2. Press the function key.

Remarks

You can make multiple selections by clicking the left mouse button and pressing the <Shift> key simultaneously or by dragging a rectangle using the left mouse button (Lasso) over the signals concerned. Please note that the rectangle must completely enclose the signals.



7 Program Creation

7.1 Blocks

In ibaLogic, a large number of function blocks are provided in a global library.

In addition to the function blocks defined in accordance with the IEC 61131-3 standard, iba also provides its own function blocks and user blocks. These are listed in the navigation area of the "Function Blocks" view.

Each function block contains an icon.

The icons have the following meaning:

-  Blocks of the IEC 61131-3
-  iba blocks
-  Functions that are available in the form of a DLL
-  Function blocks and macro blocks created by the user

The function blocks are listed in groups and sub-groups that are sorted alphabetically according to the IEC 61131-3 specification.



figure 38: Function Blocks "Global Library"

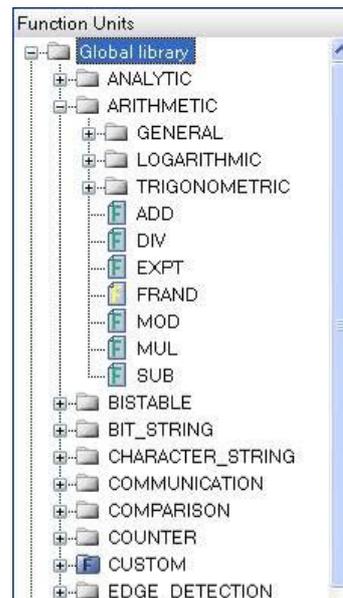


figure 39: Function Blocks "Arithmetic"

The "Global Library" also contains the folders „CUSTOM" and "NEW PROJECT".

CUSTOM

There, all global user blocks and functions being available as DLL are placed.

SPECIALS

The "Specialties" of ibaLogic are stored here. For more information, please refer to "Specials, Page 301".

7.1.1 Using Blocks

You can use all blocks from the global library and the project library in a given project.

If you wish to use a FB from the global library, you can use Drag & Drop to move it into the programming window.

You have no access to blocks that you have defined in a project of another workspace.



Important Note

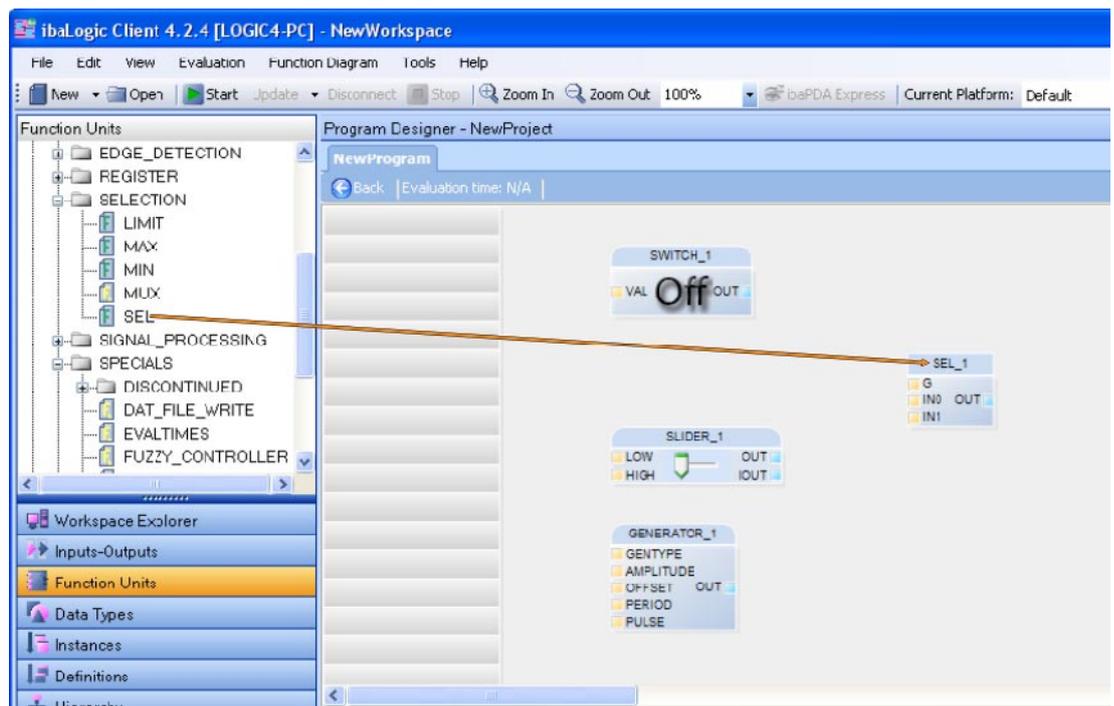
When you modify the contents of a block, all instances and the definition of the block are modified.

If you would like to modify the form, i. e. inputs and outputs or internal variables, you will be asked to specify another name for the definition. In this case, a new block type is created, the old definition and its instances remain unchanged.

However, this is only the case if more than one instance is available.

Procedure

1. Select a block that you wish to place in the programming field.
2. Drag & Drop the block selected in the "Function Block Overview" at any position that is free in the programming field.



7.1.2 Create User Blocks

You can create a block in different ways:

- In the program
- Under the project
- In the global library

7.1.2.1 In the Program

Creating an FB or MB within a program.

Procedure

1. Click with the right mouse button at any free location in the programming window.
2. In the context menu, select "New... - New Function Block" or "New... - New Macro Block". The "Create Function Block" window is displayed.
3. Create your block.
4. By pressing <OK>, the block is created if there are no syntax errors.

7.1.2.2 Under the project

Creating an FB within a project.

Procedure

1. Open the "Function Blocks" view.
2. Switch to the project folder.
3. Click in the function tree with the right mouse button on the desired project.
4. In the context menu, select "New... - New Function Block" or "New... - New Macro Block". The "Create Function Block" window is displayed.
5. Create your block.
6. By pressing <OK>, the block is created if there are no syntax errors.

7.1.2.3 In the Global Library

Creating a FB within the global library.

Procedure

1. Click with the right mouse button in the global library on the "CUSTOM" group.
2. In the context menu, select "New... - New Function Block" or "New... - New Macro Block". The "Create Function Block" window is displayed.
3. Create your block.
4. By pressing <OK>, the block is created if there are no syntax errors.



Note
Difference between Definition - Instance

For more information, please see "Instances, Page 58".

7.1.3 Managing Blocks

Copy to the global library

If you would like to use a block, which you have defined in a program or project, even in other workspaces, you must copy it to the global library.

Procedure

1. Click with the right mouse button on the FB that you would like to copy.
2. Select "Copy to the global library" in the context menu.
The block is copied into the "CUSTOM" group.



Note

A block that has been copied into the global library has the properties of the time point when it was copied. If the FB in the project path is modified, these two blocks are different.



Note

You cannot copy or move FBs from the global library to a project catalog. When using a block from the global library, it is automatically copied into the project catalog.

You can also use blocks that have been defined in another project of the same workspace. As a result, these are created automatically under the project blocks.

For more information, please refer to "Using Blocks, Page 91".

7.1.4 Exporting Blocks

If you would like to use a block, which you have defined in a database under the global group "CUSTOM" or in a project, also in another database or another program tool, then you can export this into a text file.

Procedure

1. Click with the right mouse button on the block definition under "CUSTOM" or under the project.

2. Select "Export to ST" in the context menu. The "Export" dialog box is displayed.
3. Specify the target folder and file name.



Note

If you wish to use the block exported in an ibaLogic-V4 environment, you must export the block with additional graphical information.

If you would like to use the block in another system, you should export it without additional graphical information.



Important Note

Since the implementation of the IEC standard is manufacturer-dependent, before you link the block, you must check in the external system whether it contains deviations or non-conformances with respect to the IEC standard.

7.1.5 Importing Blocks

Prerequisite

- ibaLogic is not running in the online mode.
- You have a file (Block) that can be imported.

Procedure

1. Select the "File – Import – Structured Text" menu.
The "Structured text import" window is displayed.
2. Select the file to be imported in the browser.
3. Finally, confirm with <OK>.

Check box	Explanation
Import definition of Organization unit as new	The selection imports the organization unit as a new block.

7.1.6 Removing Blocks

When you remove a FB from the program, you are deleting only one instance of it.

Procedure

1. Select a FB from the library of your project.
2. Open the context menu.
3. Select <Remove>.

Remark

A dialog box is displayed if you delete the only instance that exists. You can specify in this dialog box whether you would also like to delete the definition. Finally, confirm with <OK>. The definition is deleted.

The function or macro block is also removed from the project and is thus no longer available.

A global block definition in the "CUSTOM" group is not deleted by this method. You can do this by clicking with the right mouse button on the block in the "CUSTOM" group in the context menu "Remove" or by pressing the key.

7.2 Standard Blocks

The Appendix contains a tabular overview of all functions and function blocks that are available in ibaLogic.

7.3 Complex Function Blocks

7.3.1 DAT_FILE_WRITE (DFW Function Block)

The function with which you can save existing analog and digital signals at run time in .dat files is integrated in ibaLogic.

The .dat files have the iba data format and hence, they can be opened, analyzed and processed further with the iba tools, ibaDatCoordinator and ibaAnalyzer (e. g. extraction into a database).

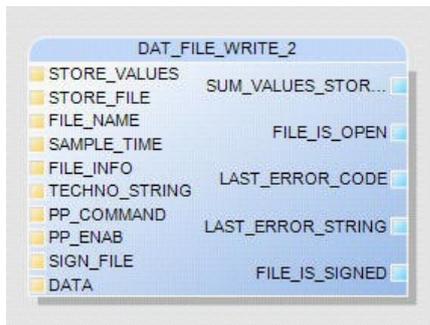


figure 40: DAT_FILE_WRITE Block

You can save either individual values or arrays of buffered data. Data of type INTEGER, REAL and BOOL is permitted. Saving additional information, such as technostring, is supported. Further information, please refer to „Buffered Mode, Page 192“.



Note

Writing to .dat files requires a license. Without a valid dongle, FILE_IS_SIGNED remains "FALSE" for recording data and hence, the files generated cannot be evaluated using ibaAnalyzer.



Note

An example of the configuration is given in the Appendix.

7.3.1.1 Function Block Edit DFW

After you have dragged the DFW block and dropped it in the program window, you can open it by double clicking on it.

The window is divided in the following tabs:

- "Arguments"
- "Graphical"

The "Graphical" tab contains the sub-tabs:

- "General Configuration"
- "Signal Configuration"

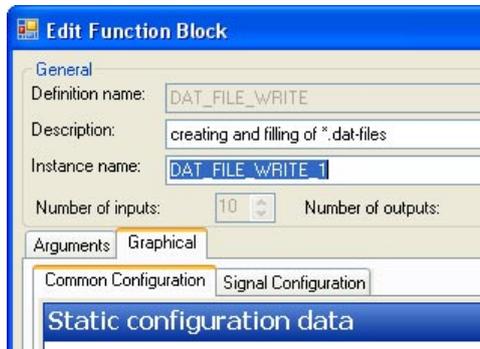


figure 41: DAT_FILE_WRITE configurator

"Arguments" tab

The "Arguments" tab displays all inputs, outputs and the associated variables and data types in a tabular view. This view is also used as an overview and to display the current values in the online mode. Do not configure any settings in this view, but switch to the "Graphical" tab. Your attention shall be drawn to any exceptions for individual properties.



Important Note

When you link an input connector, the default values and those configured in the function block are overwritten. After terminating the connection, the last value is retained.

7.3.1.2 "General Configuration" Sub-tab

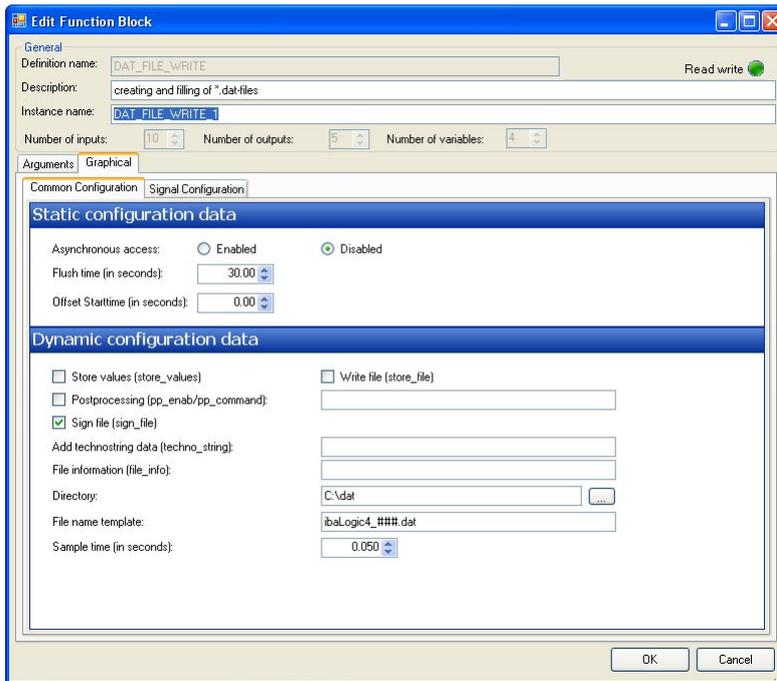


figure 42: "General Configuration" sub-tab

Asynchronous access

Disabled:

The data to be saved is buffered internally. Data is written to the hard disk when the buffer is full. If the file recording is complete, you can analyze the .dat file subsequently (using ibaAnalyzer).

Enabled:

The internal buffer is written cyclically to the hard disk after the storage cycle time configured. Hence, you can begin with the analysis while data is still being recorded. The smaller the value of the storage cycle, the earlier is data available in the ibaAnalyzer (if you have configured "automatic reload" there). The consequence is lower degree of compression and higher load on the computer or a network.

Start time offset

This time shifts the time axis backwards in time in the .dat files. The time axis in the .dat file is formed by the starting time point and the sampling time, i. e. by the number of samples and the time between the individual samples. Thus, the start time in the .dat file is the "normal start time" – start time offset.

A pre-trigger can be generated in connection with the delay in the data to be saved (using the DELAY block).

Recording starts when the trigger is activated. In order to cancel the delay in the recording, the start time offset must be set to the time delay.

Write file (store_file)

**Tip**

Do not activate this field in this tab, but do it externally via the STORE_FILE connector.

Rising edge: The .dat file with the name and path configured in the settings is created. Recording commences only when STORE_VALUES = TRUE.

Falling edge: The .dat file is closed.

 Save values (store_values)

When this value is "TRUE" and the file is open, one .dat sample is stored in each evaluation cycle.

The handling of this parameter depends on the mode of the data:

For values that are not buffered, activate this field and leave the STORE_VALUES connector unconnected. You can use STORE_FILE to control the recording.

**Note**

If you control recording of the unbuffered data using STORE_VALUES, you get an incorrect time axis. Since this is controlled in ibaAnalyzer by the number and time of the samples, switching STORE_VALUES on and off dynamically does not lead to any gaps on the time axis, but instead, the samples are arranged serially and gaps occur at the end of the .dat file.

This variable must be handled differently for buffered values. For more information, please refer to "Sub-tab "Signal configuration", Page 102".

 Post-processing (pp_enab/pp_command)

Post-processing compatible with ibaLogic V3 is available.

iba recommends using ibaDatCoordinator to realize post-processing. It provides comprehensive functions for further processing, e. g. copying files to a file server, transfer to the ibaAnalyzer for extraction to a database etc.

 Sign file (sign_file)

The file gets signed when you select "Sign File" in order to be able to use extended functions in the ibaAnalyzer. Leave this option permanently set. The associated output is also set as soon the output file gets generated. This is FALSE if the appropriate dongle is not found.

 Add technostring data (techno_string)

A technostring contains data associated with a measurement, e. g. batch number, material designations, etc. These can be evaluated specially in the ibaAnalyzer.

The technostring is saved when the file is closed.

- If you would like to use the technostring, link the connector TECHNO_STRING with a variable of type string e. g. the data received by a TCP/IP module.

❑ File information (file_info)

This is extra ASCII information that is saved when the file is closed. This information is available under "Info" when you conduct an analysis.

iba recommends that you use the FILE_INFO input connector in order to be able to modify the contents dynamically.

The info fields in the analyzer are structured as follows:

Field name: Text

If no ":" is found in the text, ibaLogic sets "UserField0" as the default value.

Multiple info fields are separated by ";".



Note

Standard info field names cannot be overwritten. Entries with this name are ignored.

Standard info field names:

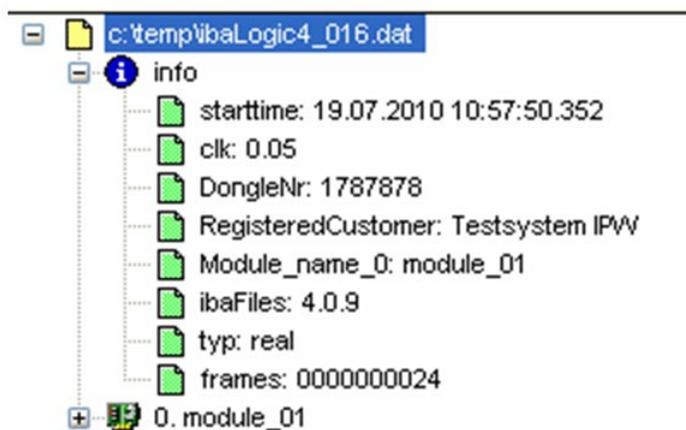


figure 43: Standard info field names

❑ Folder (Part of file_name):

- ➔ Select the drive and path by clicking on the browser button  the drive and path where the .dat files should be saved.



Important Note

You cannot work with the Windows file browser if the Runtime system is located on the PADU-S-IT platform. Specify the path and file name in the preset to the FILE_NAME connector in the "Arguments" tab or link the connector with a string variable in which you can set the path and file name dynamically. The input gets accepted when the file (STORE_FILE) is opened.

The default "C:\dat" should be used as directory. The data can then be fetched with the ibaDatCoordinator. In the ibaDatCoordinator, it can then be accessed with "\\S-IT-16-000074\RamDisk\dat", for example. Here, "S-IT-16-000074" is either the host name or the IP address of the PADU-S-IT addressed, "RamDisk" is the internal release name and "dat" the specified directory name.

Draft file name (Part of file_name):

Specification of the file name and the index. The index is incremented for each new .dat file if the path and file name specified is not modified.

The "#" character is used as a placeholder for the index. Specify multiple placeholders for multi-digit indices.

##: 0 ... 9 → 10 files

##: 00 ... 99 → 100 files

If the path is not modified, after the index values overflows, the oldest .dat files are overwritten.

Sampling time (in seconds) (sample_time)

This value is the time base for the .dat file. It defines the time in seconds between two values of a measurement signal that are saved.

In case of unbuffered values, specify the task interval for the program that contains the DFW block.

You need to enter the time period corresponding to the depth of the data array for buffered values.

For more information, please refer to "Sub-tab "Signal configuration", Page 102".



Important Note

The preparation of the signals to be measured and the DFW block must run in the same task interval. If this is not the case, the time axis in the .dat file is either stretched or compressed.

Example of sampling time

If the task in which you provide the data runs in an interval of 10 ms, you must set the sampling time in the DFW block to 0.01 sec.

If you want the values to be saved only every 50 ms, it is not sufficient only to set the sampling time to 0.05 sec. (in this case, the same data gets saved, but the X-axis contains a time period that is 5 times longer), but instead, you must specify a clock cycle at the STORE_VALUES connector that becomes "TRUE" only at every 5th cycle.

However, it is simpler to allow the block to run in a 50 ms task, and to let the STORE_VALUES connector remain static with "TRUE".



Tip

Despite correct settings of the parameters, if you see a time axis in the analysis that is too long or too short, please check whether the real task interval time (Block EVALTIMES) matches the one configured. For small cycle times it is recommended to activate the turbo mode in order to keep the task interval constant.

7.3.1.3 Sub-tab "Signal configuration"

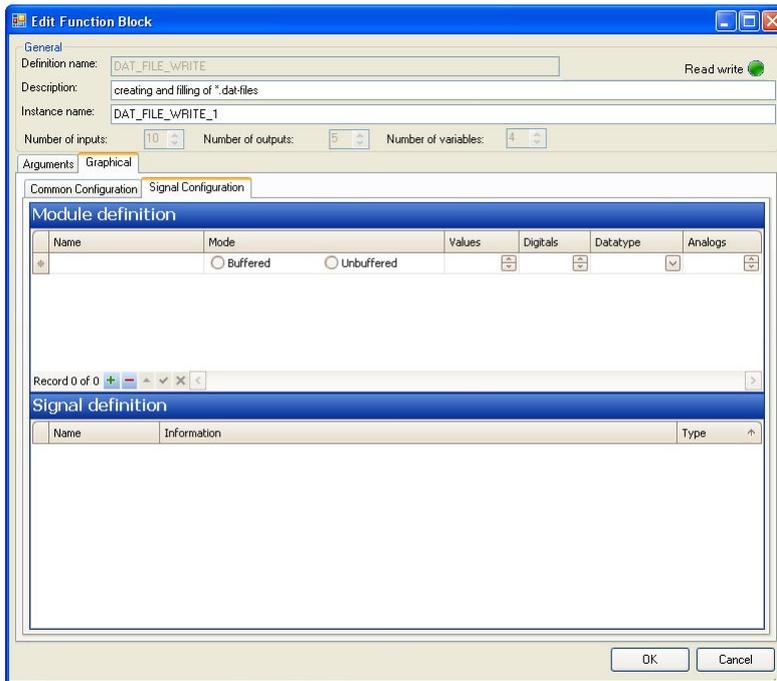


figure 44: Sub-tab "Signal configuration"

The "Signal Configuration" sub-tab contains the areas:

- Module definition:
defines the module name, module type, the number of signals and their data type
- Signal definition:
defines the signal name and the signal description



Important Note

In the present version of ibaLogic, you need to configure the modules and signals in the DFW block completely before connecting a measurement signal.

Reason: ibaLogic creates internal structure and array data types that cannot be modified once they have been used. If you modify the signal configuration subsequently, you must either remove all joiners that have been inserted automatically and rewire them or adapt all corresponding data types in the ST blocks.

In order to create a new module, an entry is provided at the end of the module list that contains a blank name field. Simply enter the module name here and configure this module. After quitting, a new entry is again generated automatically at the end of the list.

In principle, you can create as many modules as you please. Furthermore, the maximum number depends on your license.

Description of the modules:

- Name:
Module name that must conform to the IEC standard.
- Unbuffered mode:
Recording of individual values. One data sample is saved in each cycle. The value in the "Values" column is not considered.
- Buffered mode:
Recording packets. An array of data samples is saved in each cycle. The value in the "Values" column specifies the array depth, i. e. the number of samples.
For further information, please see "Buffered Mode, Page 192".
- Values:
Meaningless in the "Unbuffered" mode.
Number of samples saved per cycle in the "Buffered" mode.
- Digital values:
Number of binary signals in this module (max. 32)
- Data type of the analog values, REAL and INTEGER are permissible
- Analog values:
Number of analog values in this module (max. 32)

All signals pertaining to the module marked are displayed below the signal definition. The signals are created with the default names (Digital_nn and Analog_nn). You can edit the signal names and enter a description for each signal under "Information".



Note

You cannot modify the signal configuration as long as the "DATA" connector is connected with data. You have to break the connection if you wish to make any modifications. In the process, any joiners that have been generated automatically are removed.

Toolbar for editing the module definition record:



Symbol	Name/Tooltip	Explanation
	Append	Adds a new blank module definition record.
	Delete	Removes the module definition record selected.
	Edit	Releases the module definition record for editing.
	End Edit	The data of the modified module definition record is accepted.
	Cancel Edit	The data of the modified module definition record is not accepted. The input is canceled.
Record 1 of 1	-	Number of records

For more information, please see "Practice Examples, Page 248".

7.3.1.4 Generate Storage Structure

The simplest case of storing the .dat files generated is to specify a fixed folder and a fixed base name for the files, with ibaLogic assigning a serial number to the base name automatically.

If you need a storage structure with sub-folders and file names, which, for example, should contain the current batch number, this needs to be programmed.

Example

A new file should be created every hour. The file name should contain the hour value in the form of a name.

Implementation

The file name is formed using the current hour value. A low pulse is fed to the STORE_FILE input of the DFW when the value changes (change of hour) using the DELAY and EQ blocks. In this manner, the upcoming name is accepted for the next .dat file.

The file name is formed with the CONCAT blocks. The path name and base file name are available at the first CONCAT. The hour is appended to this and with the 2nd CONCAT the .dat extension is appended. Thus, the file name c:\iba11.dat is generated here.

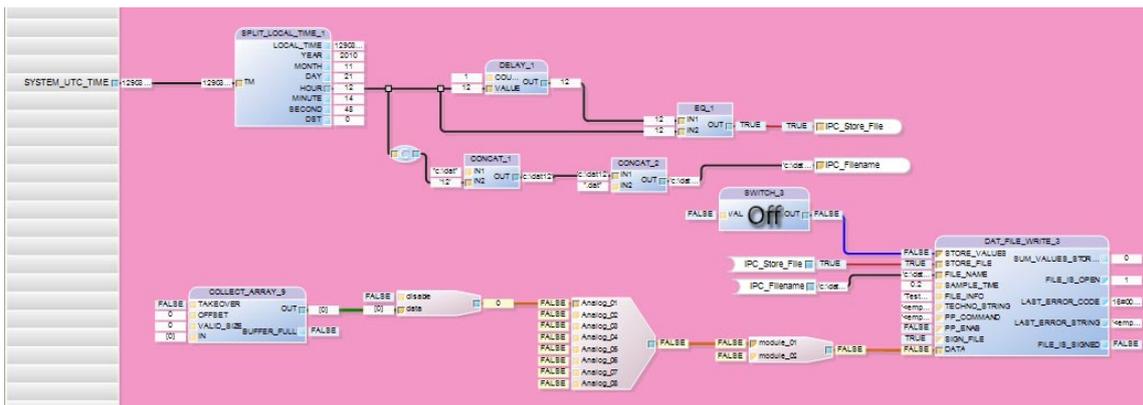


figure 45: Example: CONCAT blocks

You can also compose a unique .dat file name including the path in exactly the same way. If you specify a new path, ibaLogic creates it automatically.

7.3.2 TCPIP_SENDRECV

This block enables transmission and reception of data via TCP/IP.

The data here is raw data that is sent via TCP/IP. In this manner, all native TCP/IP protocols can be re-created.

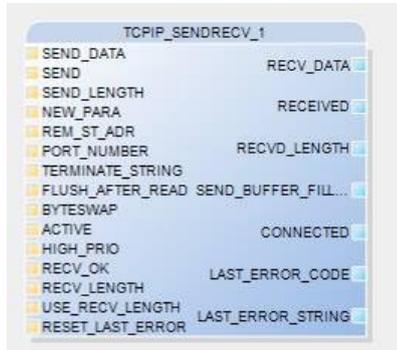


figure 46: TCPIP_SENDRECV Function block



Note

This block requires a license.



Important Note

You can configure the maximum number of parallel connections permissible in Windows XP. The value depends on the Windows configuration. You can adjust this value with the help of an entry in the Windows registry database.

For more information, please see "Number of TCP/IP connections possible, Page 318".

7.3.2.1 Inputs

Connector	Data type	Explanation
SEND_DATA	Any	Data to be transmitted The data type is ANY, i.e. it aligns itself with the interface data type, e. g. a structure, string, array.
SEND	Bool	When it is "TRUE", data available at SEND_DATA is transmitted. This input is not edge-oriented, i. e. a fixed "TRUE" at the input initiates transmission in every cycle.
SEND_LENGTH	Udint	Length of the data to be transmitted in bytes. If the value is 0, all data available is transmitted
NEW_PARA	Bool	Accept new link parameters.
REM_ST_ADR	String	IP of the link partner. This IP is required only for active link establishment, i. e. when the input ACTIVE = TRUE. If ACTIVE = FALSE, the block waits for the partner, which must specify the IP address of the PC or the PADU-S-IT. You can also enter the computer name instead of the IP address. Resolving the name may take some time during creation depending on the configuration of the Operating System.
PORT_NUMBER	Udint	When ACTIVE = TRUE: Port number of the partners. When ACTIVE = FALSE: Own port number
TERMINATE_STRING	Udint	Strings are terminated with a NULL byte. This input is evaluated only when strings are available at the SEND_DATA input.
FLUSH_AFTER_READ	Bool	Deletes the receive buffer after reading the data.
BYTESWAP	Int	= 1: Swap based on data type (AB CDEF → BA FEDC) = 2: Swap 2 Bytes respectively (ABCD → BADC) = 4: Swap 4 Bytes respectively (ABCD → DCBA)
ACTIVE	Bool	TRUE: (ibaLogic is the TCP/IP client) The block attempts to establish a link to the IP and the PORT, which are specified in REM_ST_ADR and PORT_NUMBER respectively. FALSE: (ibaLogic is the TCP/IP server) It waits for incoming connections at the port number configured. The IP address is not evaluated.
HIGH_PRIO	Bool	The data is fetched with a higher priority from the Windows network buffer and written in the input buffer. NOTE By default, this function should be set to "FALSE".
RECV_OK	Bool	TRUE: Data received is OK, and the input buffer can be filled with the new data. FALSE: The data last received remains until the input is triggered again.
RECV_LENGTH	Udint	Defines the telegram length in bytes. This input is evaluated only if the input, USE_RECV_LENGTH = TRUE
USE_RECV_LENGTH	Bool	TRUE: If the input buffer is larger than the RECV_LENGTH configured, there is only one telegram with the length configured at the output, RECV_DATA. FALSE: At the output, RECV_DATA, there is one telegram with the maximum size of the data type available at the output, RECV_DATA.
RESET_LAST_ERROR	Bool	Resets the error outputs

7.3.2.2 Outputs

Connector	Data type	Explanation
RECV_DATA	Any	Received data. The data type aligns itself with the interface data type, e. g. a structure, string, array.
RECEIVED	Bool	This output is TRUE as soon as any data is received
RECVD_LENGTH	Udint	Length of the data received in bytes.
SEND_BUFFER_FILLED	Bool	TRUE when the internal transmit buffer is full. i. e. the block cannot transmit data fast enough
LAST_ERROR_CODE	Dword	Last error message as a DWORD in hex format
LAST_ERROR_STRING	String	Last error message as clear text

Example of a Send - Receive procedure

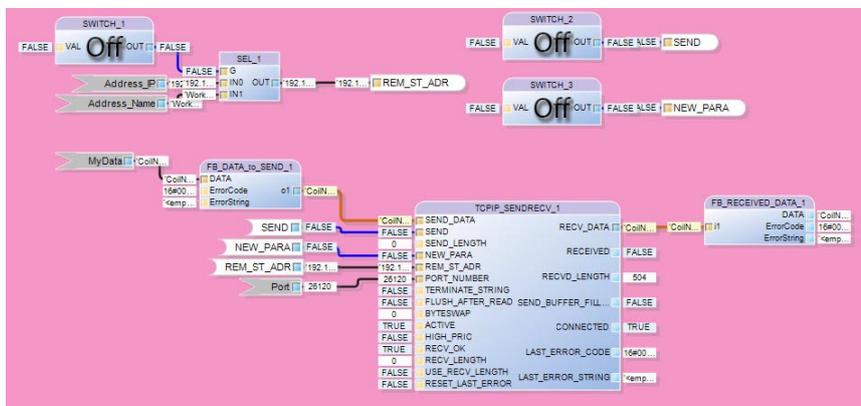


figure 47: Example of a Send - Receive procedure

In this example, the data to be transmitted is available in the form of a structure. The data is transmitted with the help of a manual trigger.

There is a switch at NEW_PARA, in order to re-establish the link for test purposes or in case of modification in the link parameters, if required.

This block is passive and waits until a link has been established by the communication partner.

The data received arrive at RECV_DATA. The input RECV_OK is set permanently to "TRUE", since the data does not have to be buffered intermediately, as they can be well processed within the task cycle time configured.

7.3.3 PIDT1_CONTROL

Universal PIDT1 controller that can be switched to operating modes as a P, I, PI or PIDT1 controller.

- Set the starting value of the integrator
- Hold the instantaneous value of the integrator
- Pre-control value WP
- Controller limits LL and LU
- Proportional coefficient KP

- Reset time TN
- Control direction reversible
- Indication when the limits configured are reached
- Display of the error signal
- Display of separate P, I and DT1 controller outputs



figure 48: PBDT1_CONTROL Function block

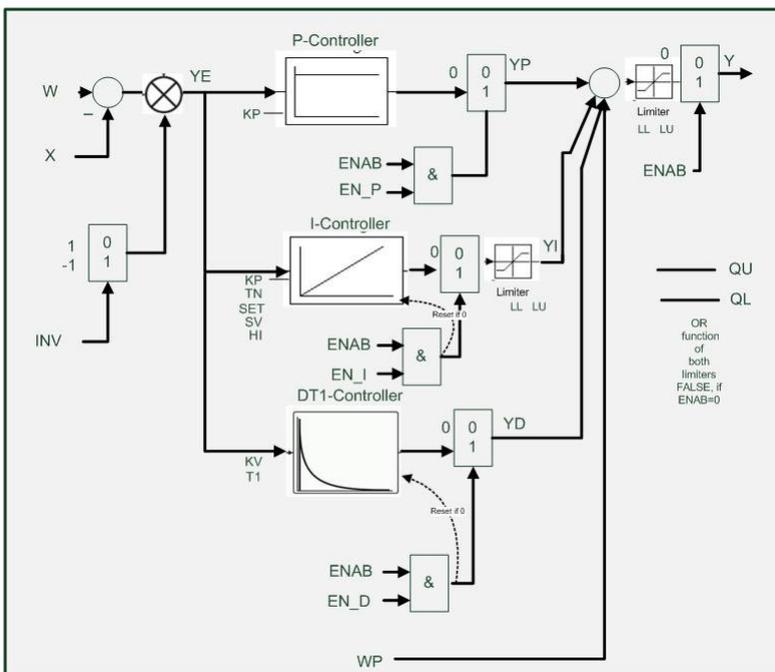


figure 49: Block diagram of a universal PIDT1 controller

7.3.3.1 Inputs

Connector	Data Type	Explanation
W	Lreal	Set-point value
X	Lreal	Actual value
WP	Lreal	Pre-control value
LL	Lreal	Lower limit value (valid for Y and YI)
LU	Lreal	Upper limit value (valid for Y and YI)
SV	Lreal	Set value for the integrator is accepted with SET
KP	Lreal	P gain
TN	Time	Reset time
KV	Lreal	D gain
T1	Time	D time constant
ENAB	Bool	Controller release
INV	Bool	Invert the sign of the control deviation
EN_P	Bool	Activate the P controller
EN_I	Bool	Activate the I controller
SET	Bool	Set the integrator with the value SV
HI	Bool	Stop the integrator
EN_D	Bool	Activate the D controller

7.3.3.2 Outputs

Connector	Data type	Explanation
Y	Lreal	Control value = $Y_P + Y_I + Y_D + W_P$
YE	Lreal	Control deviation = $W - X$
Y_P	Lreal	Output value of P controller = $K_P * Y_E$
Y_I	Lreal	Output value of I controller = $Y_{I,n-1} + K_P * Y_E * T_a / T_N$
Y_D	Lreal	Output value of D controller = $\alpha * Y_{D,n-1} + \alpha * K_V * \Delta Y_E$ $\alpha = 1 / (1 + T_a / T_1)$ $\Delta Y_E = (Y_E - Y_{E,n-1})$
QL	Bool	Lower limit value reached
QU	Bool	Upper limit value reached

7.3.3.3 Details / Signal trends

The various signal trends of the individual controller components are illustrated in the following diagrams.

Control value Y:

The control value Y is the sum of the P, I and D components and the pre-control value WP.

If the input, ENAB (controller release) is not set, the control value is always 0.0.

Input WP, pre-control value:

This input is added to the output Y.

Input LL/LU, lower / upper limit value:



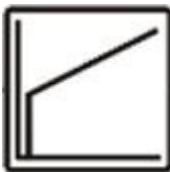
Note

The total output Y, as well as YI are limited.

The outputs, QL and QU, accordingly take up the value TRUE.

The following controllers are used in practice:

PI controller



PD controller



PID controller



The P, I and DT1 components are given special consideration in the following.

7.3.3.4 P component: (Parameter: KP, EN_P)

The P component of the controller is calculated as $KP \cdot YE$. The value is fed to the output value only when EN_P is set.

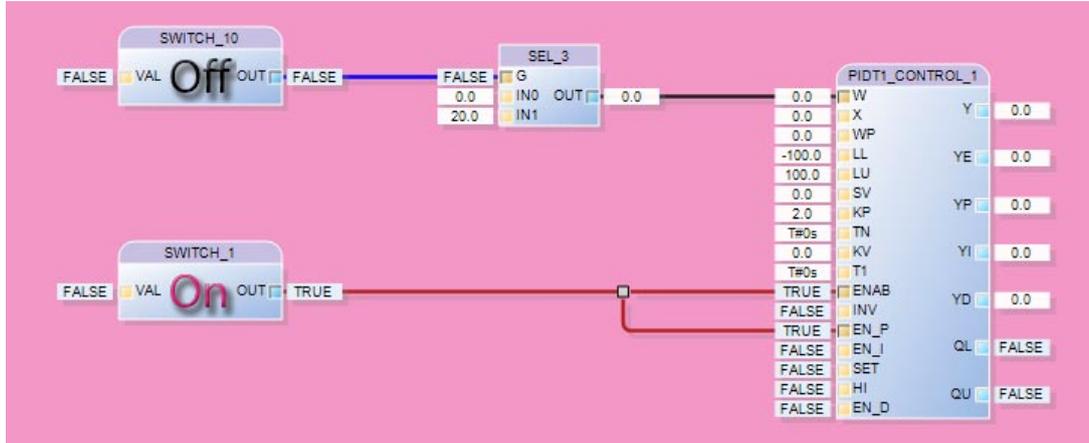


figure 50: P component

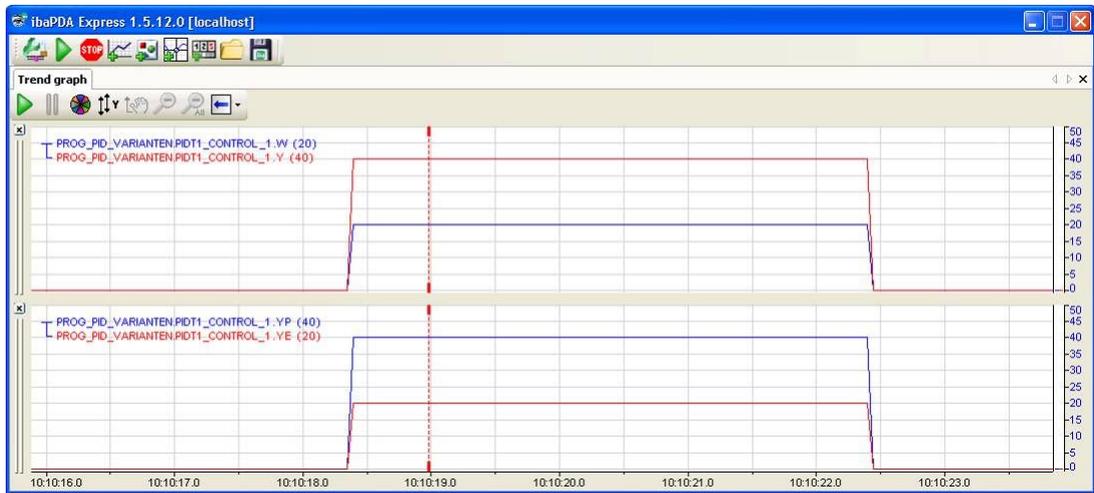


figure 51: P component

7.3.3.5 I component: (Parameters KP, TN, SET, SV, HI and EN_I)

The I component of the controller is calculated as $YI_n := YI_{n-1} + KP * YE * Ta/TN$ (Ta = Task time).

This component can be set to the value of the input SET with the input SV. A value of "TRUE" at the HI input stops the integrator. The value is fed to the output value only when EN_I is set.

Correlations

$$TN = KP * Ta = KP / KI$$

Example 1

KP = 1.0

TN = 1 s

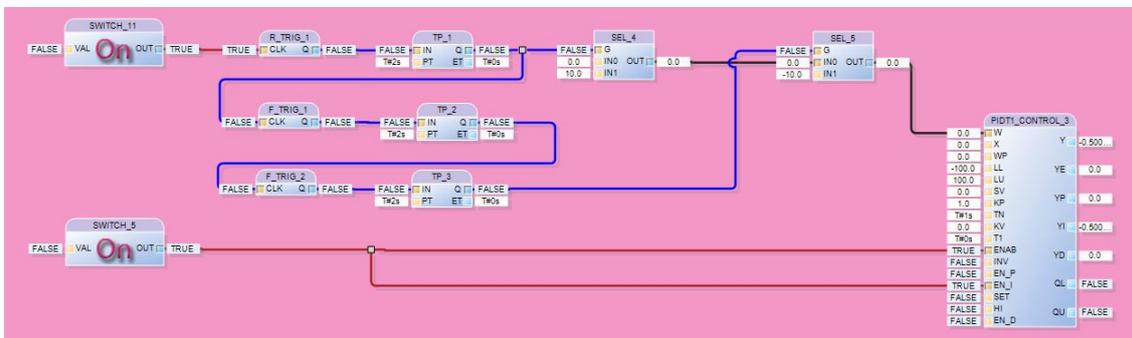


figure 52: I component

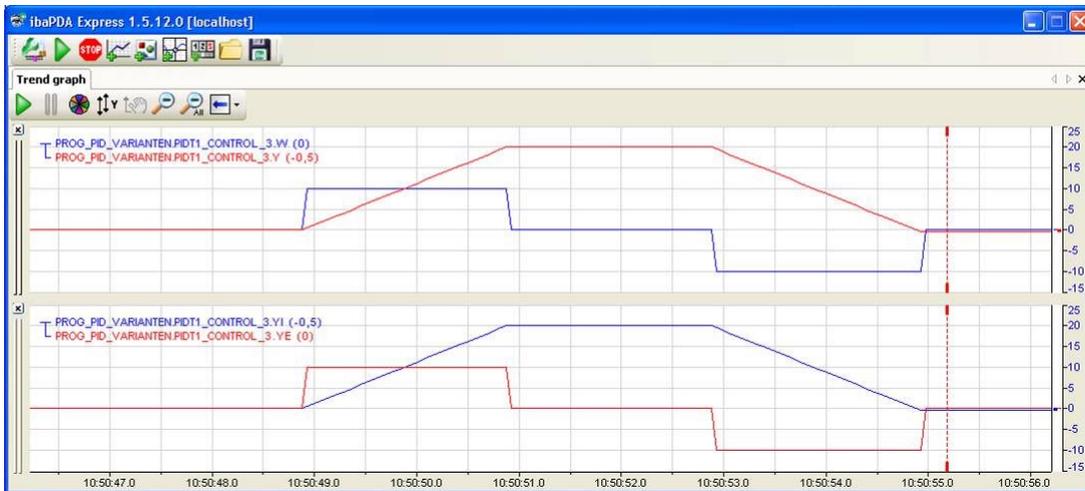


figure 53: I component

7.3.3.6 DT1 component: (Parameters KV,T1 and EN_D)

The DT1 component of the controller is obtained as $YD := \alpha * YD_{n-1} + \alpha * KV * \Delta YE$

$$\alpha = 1 / (1 + Ta / T1) \quad \Delta YE = (YE - YE_{n-1}). \quad (Ta = \text{Task time})$$

The value is fed to the output value only when EN_D is set.

Example 1

KV = 0.5

T1 = 1 s

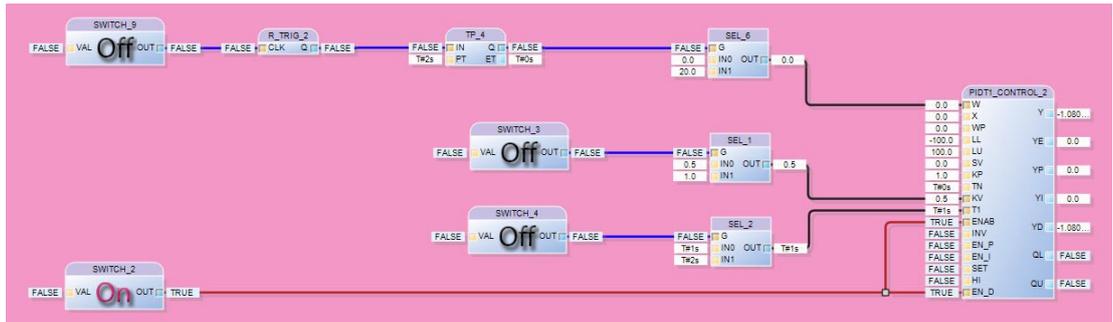


figure 54: DT1 component

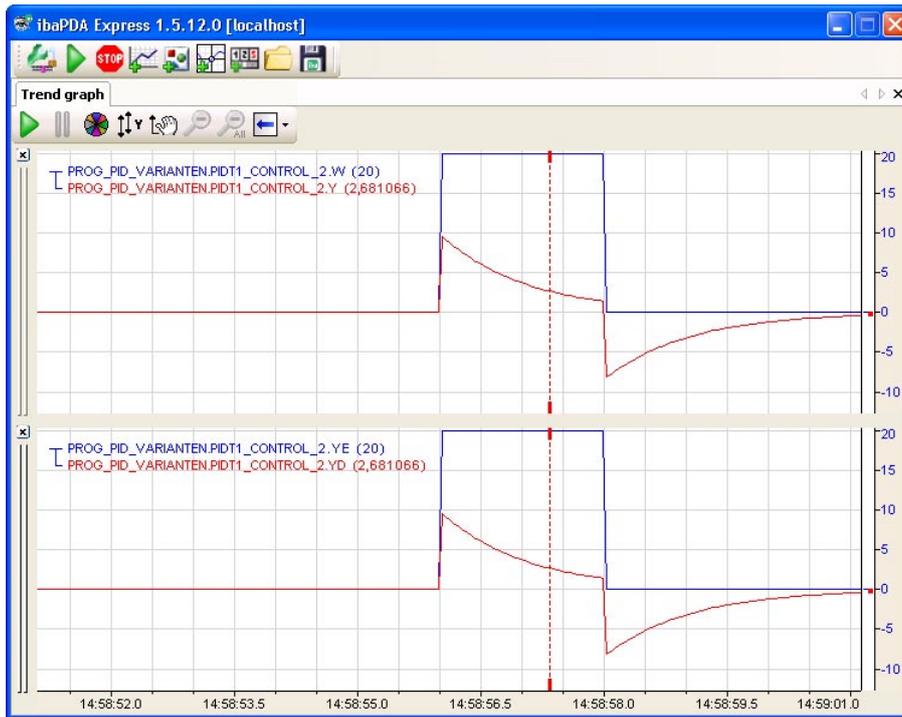


figure 55: DT1 component

Example 2

KV = 1

T1 = 2 s

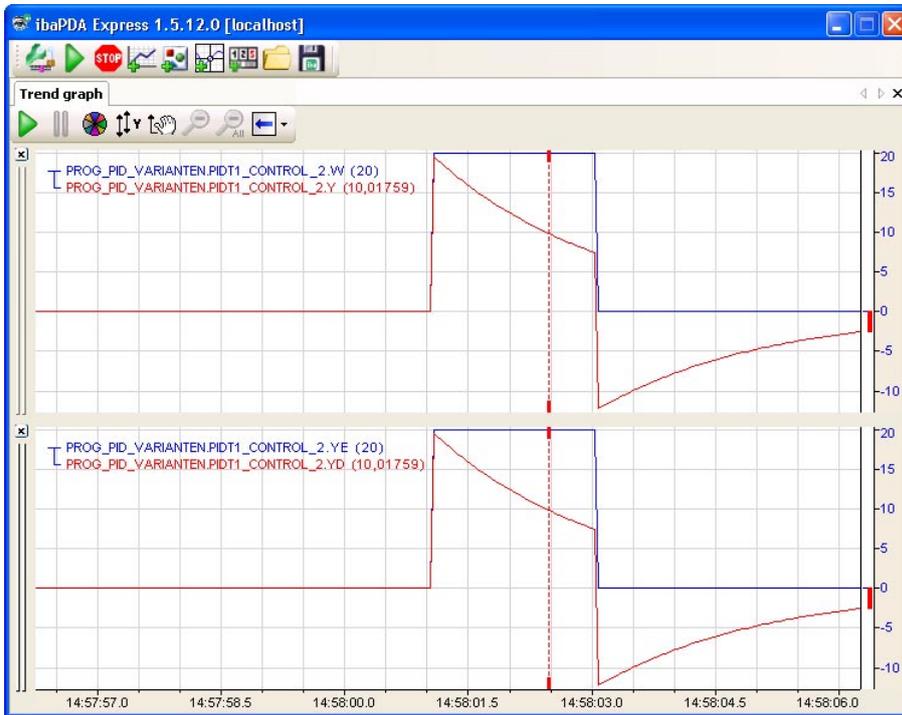


figure 56: DT1 component

7.3.3.7 PIDT1 component – Total response

Example 1

Example of the complete PIDT1 controller with signal trends.

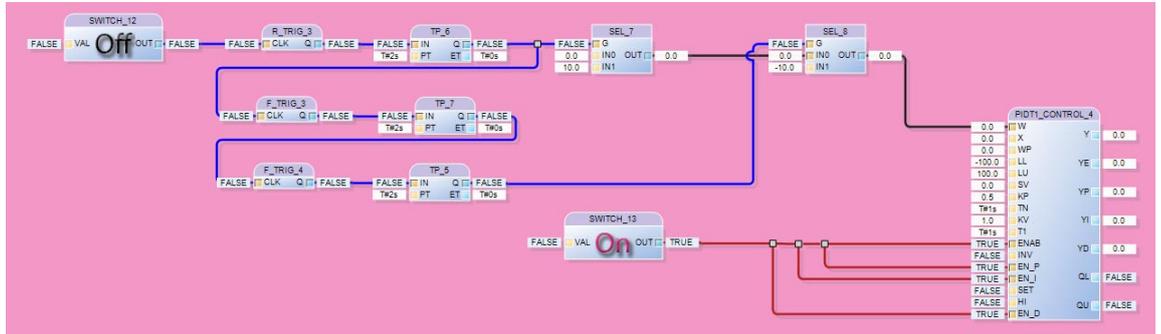


figure 57: PIDT1 controller with signal trends

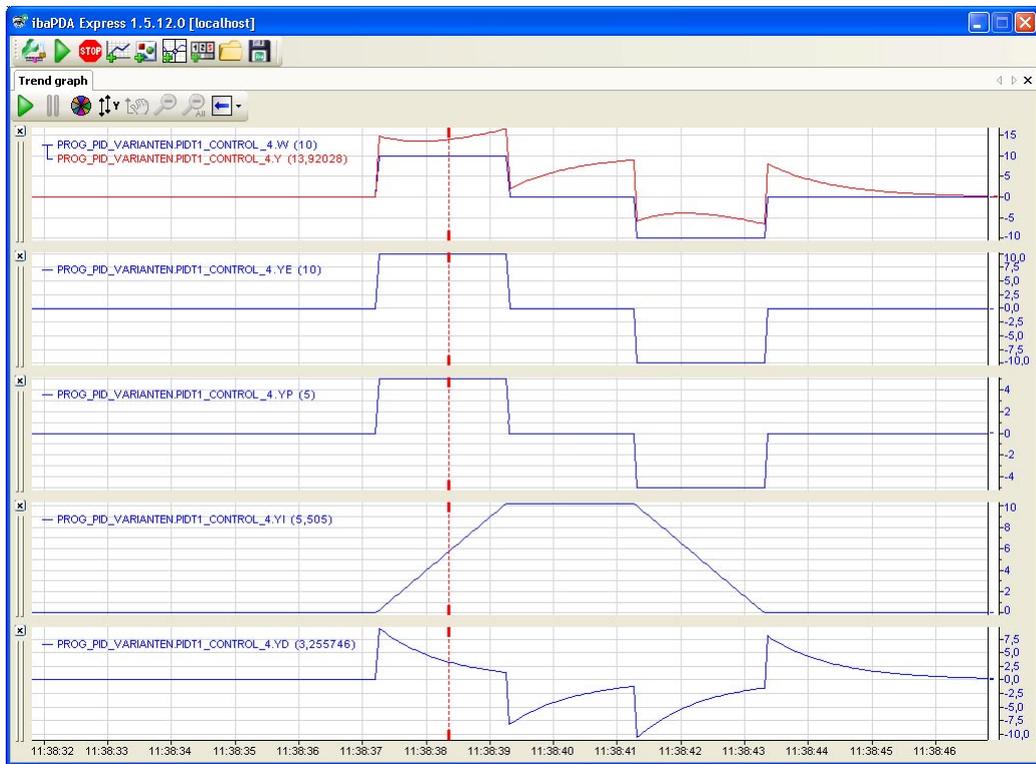


figure 58: PIDT1 controller with signal trends

7.3.4 RAMP

Ramp block with 2 different ramps: Manual and automatic mode

- Set-point value limit
- Start up the new set-point value via the ramp
- Set the set-point value
- Indication when the limit values are exceeded



figure 59: RAMP Function block

7.3.4.1 Inputs

Connector	Data Type	Meaning / Usage
X	Lreal	Input value (Set-point)
LL	Lreal	Lower limit value
LU	Lreal	Upper limit value
SV	Lreal	Set value, output is set to this value with SET
RM	Lreal	Manual ramp (1/s), valid for CD and CU
RA	Lreal	Automatic ramp (1/s), valid for CF
CD	Bool	Ramp falling (manual ramp control)
CU	Bool	Ramp rising (manual ramp control)
CF	Bool	Ramp as per the input value (automatic ramp control), has precedence over CD and CU
SET	Bool	Set output value to SV

7.3.4.2 Outputs

Connector	Data Type	Meaning / Usage
Y	Lreal	Output value; $Y_n = Y_{n-1} + UR$ r = ramp used
UR	Lreal	Ramp used (1/s)
QE	Bool	Output value = Input value
QL	Bool	Lower limit value reached
QU	Bool	Upper limit value reached

7.3.4.3 Example

The inputs CD, CU and CF control the ramps. If none of the inputs is active, the last output value is fixed. The output UR then displays the ramp used as the value 0.

If the input CD is active, regardless of the input value, the current output value at the manual ramp is scaled down to a maximum of the lower limit.

If the input CU is active, regardless of the input value, the current output value is raised via the manual ramp up to a maximum of the upper limit.

If both CD and CU are active simultaneously, UR is set to 0. The output value does not change.

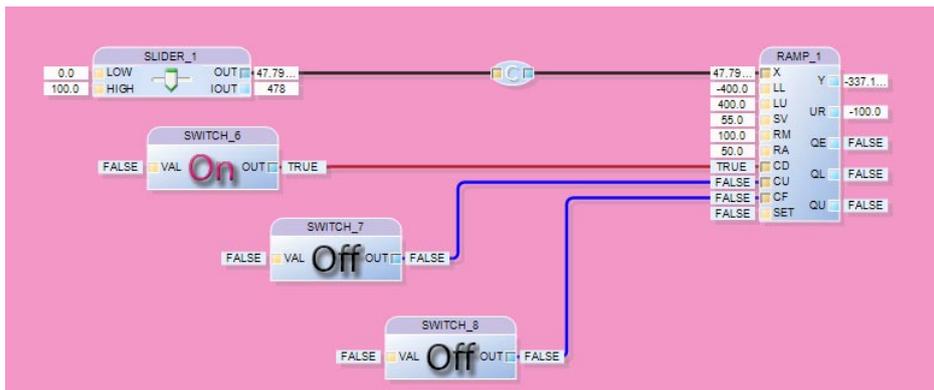


figure 60: Controlling ramps

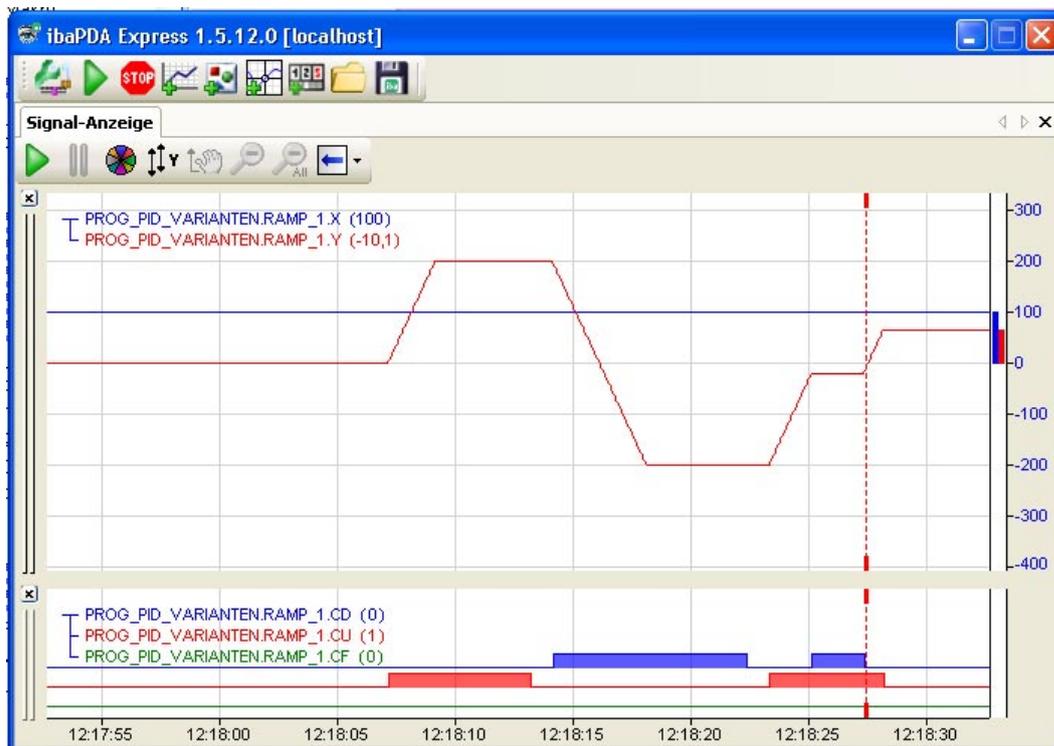


figure 61: Controlling ramps

If the input CF is active, the output value tracks the input value via the automatic ramp. If the input value exceeds the limit, the output value changes in line with the ramp only up to the limits.

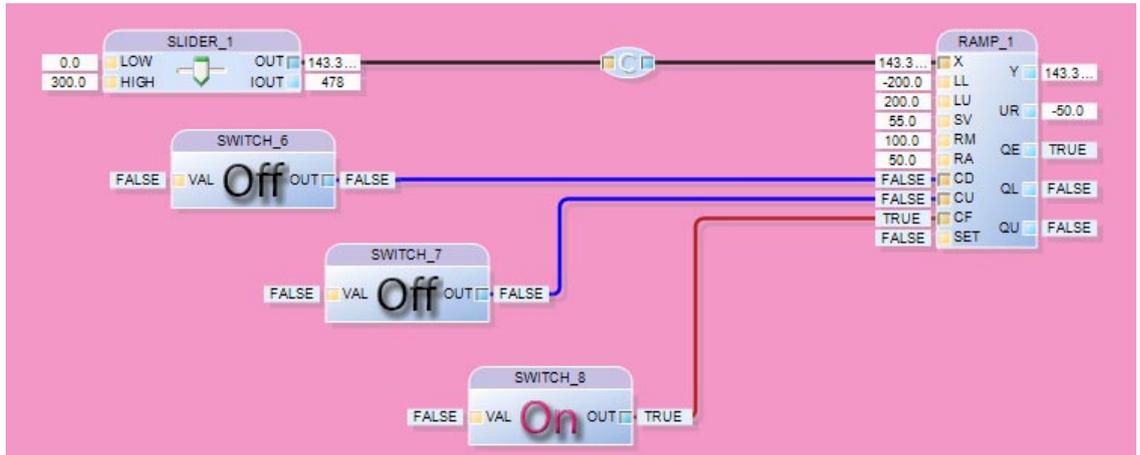


figure 62: Controlling ramps



figure 63: Controlling ramps

If CF is set, the inputs CD and CU do not have any effect.

7.3.5 FUZZY_CONTROLLER

Fuzzy logic is a theory that, above all, has been developed for modeling uncertainties and fuzziness of colloquial descriptions. For example, you can use it to capture the fuzziness of specifications such as "a little", "quite" or "considerable" mathematically in models.

Fuzzy logic is beneficial particularly when a precise logical or mathematical description for a process is not available, cannot be created or the correlations are too complex, but, for example, intuitive know-how of an operator is available.

Fuzzy logic is based on fuzzy quantities and so-called associative functions, which map objects to fuzzy quantities and to compatible logical operations on these quantities and their inference. Moreover, in the case of technical applications, methods must be considered for the conversion of specifications and correlations in fuzzy logic.

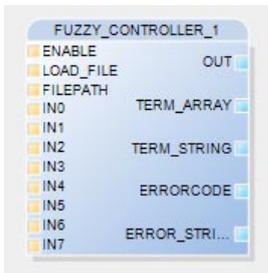


figure 64: FUZZY_CONTROLLER Function block

ibaLogic provides the FUZZY_CONTROLLER function block for control using fuzzy logic.

Principle of Operation

The fuzzy controller obtains an output value based on a set of parameters that it takes from a parameter file related to the application. With the commencement of the evaluation with connector ENABLE = TRUE, the parameters from the file whose path specified to the connector via FILEPATH, and evaluation is done in the same evaluation in ibaLogic for obtaining the output value with the new data loaded.

The parameter file can be reloaded via the LOAD_FILE control signal even during the current evaluation, without interrupting the output value evaluation or data acquisition in ibaLogic. The output value can be derived from up to 8 input values IN0 ... IN7, and is used as a set-point value at the OUT connector or as a control variable of a process.

7.3.5.1 Inputs

Connector	Data Type	Explanation
ENABLE	Bool	Start the evaluation procedure when the input is "TRUE". Data from the parameter file is accepted with the rising edge.
LOAD_FILE	Bool	Accept the data in the parameter file with the rising edge of the signal during the evaluation time.
FILEPATH	String	Path specification for the data of the parameter file
IN0..IN7	Lreal	Input data, on the basis of which the output value is evaluated.

7.3.5.2 Outputs

Connector	Data type	Explanation
OUT	Lreal	Output value of the fuzzy controller; the output = 0,0 in case of an error or if enable = "FALSE"
TERM_ARRAY	Lreal (0..8)	Level of association $\mu(x)$ of the individual linguistic terms from which the output value is formed
TERM_STRING	String	Specification of the current linguistic terms from which the output value and the percentage of the level of association are formed.
ERROR_CODE	Int	Output of the error code
ERROR_STRING	String	Text output of a brief error message.

Handling

Configuring and customizing is performed using the "ParamFuzzyTool.exe" application. You can get this upon request from the iba Support and contact, Page 330.

The function block of the fuzzy controller is located under "Function blocks - SPECIALS".

The path of the parameter file is provided as a string to the FILEPATH connector. With the rising edge of the Boolean signal at the ENABLE connector, data from the parameter file is accepted and the evaluation of the output value commences promptly. As long as the Enable signal is "TRUE", the output value is evaluated using the LREAL values available at the connectors IN0 ... IN7. The default value of the ENABLE connector can be defined with "TRUE". This results in permanent evaluation of the output value.

The connector of the LOAD_FILE Boolean signal at the function block of the fuzzy controller offers the option of loading a new parameter file while an evaluation is being performed. The rising edge of the LOAD_FILE signal causes the data in the fuzzy block to be accepted as the new set of parameters to form the basis of the output value evaluation.

7.4 User-specific Function Blocks

ibaLogic has a library of pre-defined function blocks. Nonetheless, it may be necessary to define your own function blocks for a more efficient solution to your problem. There are three types of user-specific function blocks:

- Function blocks that are created in ibaLogic using the high-level programming language, Structured Text (ST).
- It is also possible to combine existing graphics programming to macros.
- Function blocks that are created external to ibaLogic in a high-level language (C++, others on request e.g.: FORTRAN) and are integrated as DLL into ibaLogic.

After they are created, all these function blocks are treated like standard function blocks.

7.4.1 Function Blocks

In order to create a new function block, position the mouse pointer to a free location in the program window and select "New - New Function Block..." in the context menu. The "Create Function Block" dialog box is displayed.

The fields for the name and the table for defining the inputs, outputs and internal variables are located at the top of the box.

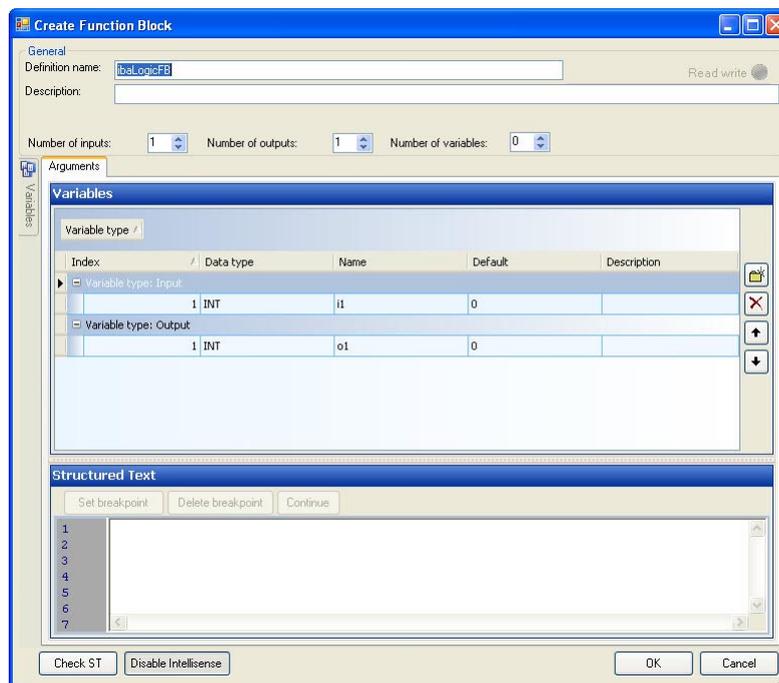


figure 65: "Create Function Block" dialog box

7.4.1.1 General Settings

Definition name

The definition name is the name with which the block is saved in the block folder. The instance name is formed from this name by appending an index.



Note on difference between Definition - Instance

For more information, please see "Instances, Page 58".



Note

iba recommends that you use different prefixes for function blocks and macros, e.g. "FB_" and "MB_". You can configure the settings under the "Tools – Options – Function Blocks" menu.

Similarly, you can also find the default values for the names and data types of the variables, and iba recommends using the names "i", "o", "io" and "v". You can configure data types based on your requirements.

Instance name

The instance name is not displayed during creation. It is displayed only when you retrieve a block to use it in a project.

Description

You can describe the block function in more detail in this field. This description is displayed as a tooltip when you move the mouse pointer over the function block name in the library.

Number of Inputs / Outputs / Variables

You specify the number of variables used here. One line is created in the table for each variable.

Variables

The list of variables is available either as a tree or as a table for display.

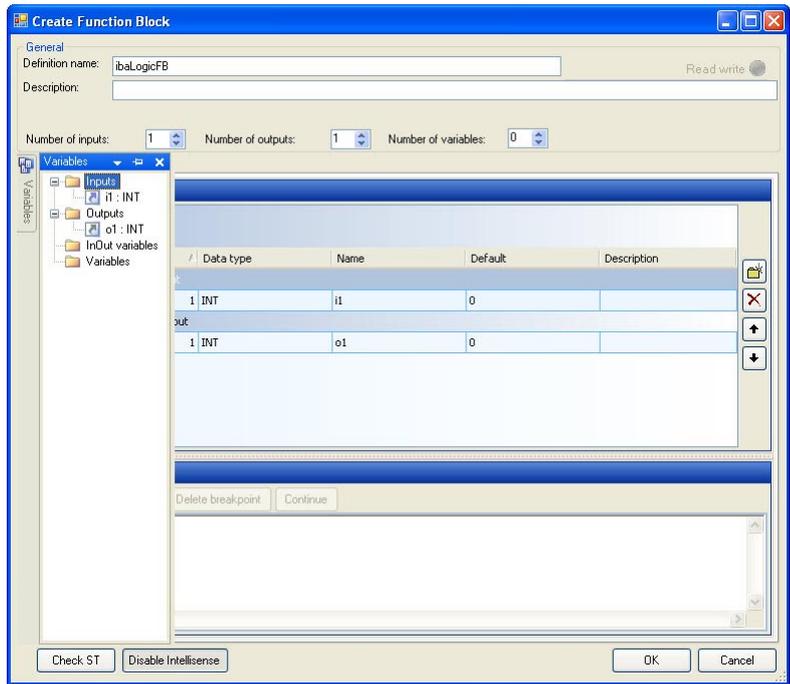


figure 66: "Create Function Block" dialog box with the list of variables

➡ You can open the tree by clicking on the "Variables" tab to the left of the table.

This view is hidden since you can configure and customize the variables only in the variables table.

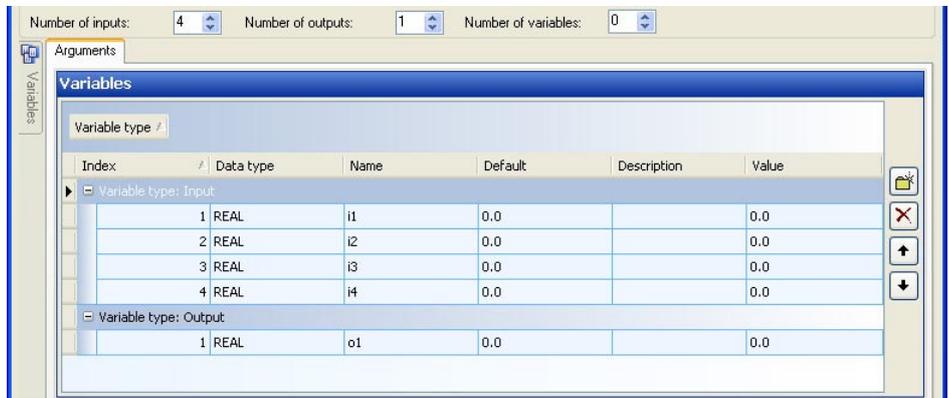


figure 67: Function block variables

Variable

Column	Explanation
Index	Each variable type begins with index 1.
Data Type	Selection box for accepting a defined variable type. You can also create new user types here. You can find the default value under the "Tools Options" menu.
Name	Default value consists of prefix and index. You can, however, also edit a new name.
Default value	Please note that the notation of the values depends on the data type. For more information, please refer to "Syntax Description of Structured Text, Page 128". The value is assigned to the variable provided that no link is connected (for input variables) or there is no assignment within the block code.
Description	Text field that is displayed as a tooltip in the block folder.
Value	<p>In the case of arrays and structures, the current values of the variables displays only the first element (only in the online mode).</p> <hr/> <p> Tip This value can be modified manually, but it is re-evaluated in the next cycle and hence, if required, overwritten.</p> <hr/> <p> Important Note If you remove a link to the input connector in the online mode, the last value remains.</p>

Using the buttons on the right, you can change the sequence of the variables, add and delete new variables at the position marked.

Icon	Explanation
	Add an element.
	Delete an element.
	Interchange the elements.

7.4.2 Structured Text Editor

You can define the functionality of a function block using the Structured Text Editor.

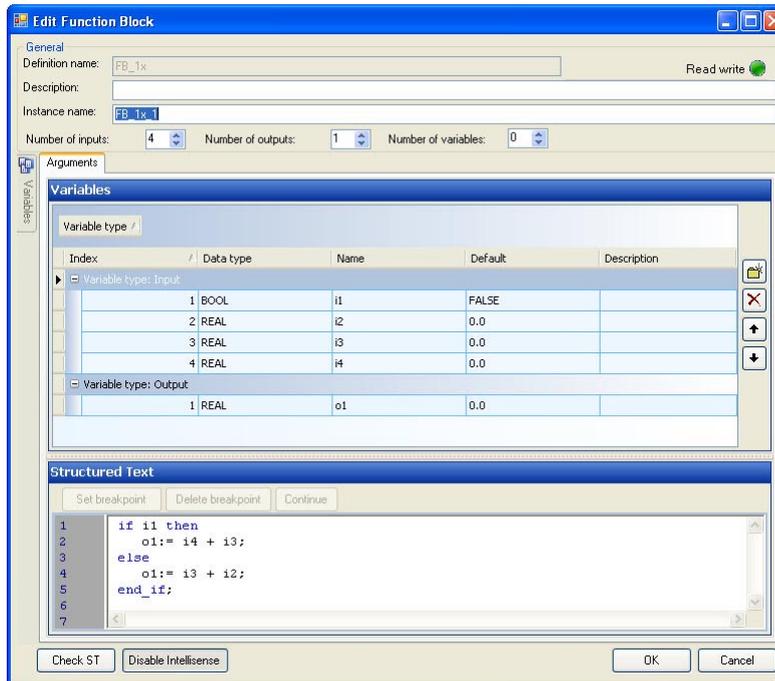


figure 68: Structured Text Editor

The following buttons are located above and below the text input field:

button	Explanation
Set a breakpoint (active only in the online mode)	By clicking on this button, program code execution stops at the insert text position of the mouse cursor.
Delete breakpoint (active only in the online mode)	It is used to remove the breakpoints at the insert text position of the mouse cursor.
Continue (active only in the online mode)	Program execution is continued. Program execution is stopped again at the breakpoint in the next cycle.
Check ST	Syntax test of the code entered, without compiling it
Enable / Disable Intellisense	Enable or disable the Intellisense resource

⚠ DANGER

Danger by using functions in the online mode!

We dissuade you strongly from using these functions (Set breakpoint, delete breakpoint, continue), if you are using outputs for control and regulation functions in ibaLogic in the online mode, since there is a risk of personal injury and damage to property that could result by doing so (see "Time behavior, Page 230").

7.4.2.1 IntelliSense

IntelliSense is a resource for completing entries automatically. It provides additional information and selection options to the programmer to facilitate the completion of data entry.

During the creation of blocks, also new variables are automatically added to IntelliSense.

In particular, it simplifies working with structures considerably, since with structure variables, after entering the "." separator, all elements defined are provided immediately for selection.



figure 69: Structured Text editor with IntelliSense enabled

Example: The IntelliSense selection window appears by entering "IF". You can enter the choice highlighted with <Return>.

Statements such as IF..THEN..ELSE / WHILE.../ REPEAT... are provided only after you enter the first word, and can, hence, be taken over completely at this stage as a framework.

You can make the selection using <Cursor up> or <Cursor down>, and accept the entry with <Tab>.

7.4.2.2 Syntax Description of Structured Text

For example, Structured text can look like the following:

```

1 (* Difference between i2 and i1 *)
2 Difference := i2 - i1;
3
4 (* Mean value calculation *)
5 Mean value := (i1 + i2) / 2.0;

```

figure 70: Syntax of Structured Text

Notations:

- Comments are enclosed in "(" and ")".
- Statements must be terminated with a semicolon.
- The value of the result must be written on the left of ":=".
- Expressions consist of operators and operands.



Important Note

Apart from the operators and statements described in the following, you can also call up some functions that are available graphically as a block, even from within the ST. You will find notes on whether and how these can be in the ST in the description of the functions in the section "Standard Function Blocks, Page 280". There, for each block with the keyword "ST": a note has been provided regarding its usability in ST.

7.4.2.3 Operators

List of the operators sorted by precedence:

Operator	Example	Value in the example	Description	Priority
()	(2+3) * (4+5)	45	Brackets	Highest
**	3**4	81	Exponentiation	
-	-10	-10	Negation	
NOT		NOT TRUE	Logical negation	
*	10**3	30	Multiplication	
/	6/2	3	Division	
MOD	17 MOD 10	7	Modulus (Division remainder)	
+	2+3	5	Addition	
-	4-2	2	Subtraction	
<, >, <=, >=	4 > 12	FALSE	Comparison	
=	T#26h = T#1d2h	TRUE	Equality	
<>	8 <> 16	TRUE	Inequality	
&, AND	TRUE & FALSE	FALSE	Boolean AND	
XOR	TRUE XOR FALSE	TRUE	Boolean Exclusive Or	
OR	TRUE OR FALSE	TRUE	Boolean Or	Lowest

7.4.2.4 Statements

Key-word	Example	Description
;	;	Blank statement
:=	Var1 := 12;	Assigning the value 12 to the variable name given on the left.
f(i1, i2, ...)	o1 := concat (iDir, iFile, v1);	Function call, see also the description of the function blocks
IF	IF i1 < i2 THEN o1 := 1; [ELSIF i1 =i2 THEN o1 := 2;] ELSE o1 := 3; END IF;	Conditional statement. The condition is a Boolean expression (that yields the result "TRUE" or "FALSE") Optional extensions are given in square brackets [...].
CASE	CASE i1 OF 1: o1:=3; 2: o1:=4; 3,4,5: o1 := 5; o2 := 6; 11..15: o1:= 11; [ELSE o1 := 0; o2 := 0;] END_CASE;	Case statement. The argument "i1" is an expression of type ANY_INT or ENUM. There are one or more statements per case. A case can have several integers (3,4,5) or enumerators or ranges of integers (10...15). The ELSE branch is optional. Optional extensions are given in square brackets [...].

Key-word	Example	Description
FOR	<pre>FLAG := FALSE; FOR ix:= 1 TO 100 [BY 2] DO IF o1[ix] = iy THEN FLAG := TRUE; EXIT; END_IF; END_FOR; IF FLAG THEN (* found*)</pre>	<p>Unconditional loop (Iteration).</p> <p>The step (BY xx) is optional. If it is not present, the step is 1.</p> <p>The type of the loop variable is ANY_INT and it should not be modified within the loop.</p> <p>Optional extensions are given in square brackets [...].</p> <p>Attention</p> <p>There is a risk of endless (infinite) loops</p>
WHILE	<pre>WHILE i1 > 1 DO o1 := o1/2; END_WHILE;</pre>	<p>Conditional loop</p> <p>Attention</p> <p>There is a risk of endless (infinite) loops</p>
REPEAT	<pre>REPEAT o1:= o1 * i1; UNTIL o1 > 10000 END_REPEAT;</pre>	<p>Conditional loop</p> <p>The difference to WHILE is: The loop is executed at least once, even if the condition is not met right at the beginning.</p> <p>Attention</p> <p>There is a risk of endless (infinite) loops</p>
EXIT	<pre>FLAG := FALSE; FOR ix:= 1 TO 100 [BY 2] DO IF o1[ix] = iy THEN FLAG := TRUE; EXIT; END_IF; END_FOR; IF FLAG THEN (* found*)</pre>	<p>Premature termination of a FOR, WHILE or REPEAT loop.</p> <p>The first statement is executed after the next end of the loop, i. e. with nested loops, execution continues with the next higher level.</p> <p>Optional extensions are given in square brackets [...].</p>
RETURN	<pre>oFLAG := FALSE; FOR ix:= 1 TO 100 [BY 2] DO IF o1[ix] = iy THEN oFLAG := TRUE; RETURN; END_IF; END_FOR;</pre>	<p>Return statement, premature termination of the function block.</p> <p>Example: If the value iy is contained in the array ix, the result is TRUE, otherwise it is FALSE.</p> <p>Optional extensions are given in square brackets [...].</p>
ARRAY access	<pre><ArrayType>[index,...] o1 := iArray[0]; o1 := iArray[0,0,...]; o1 := iArray[0][0];</pre>	<p>The indices are given in square brackets</p> <p>Access to a 1-dimensional array</p> <p>Access to an n-dimensional array</p> <p>Access to a nested array</p> <p>For more information, please refer to "ARRAY TYPE Group, Page 148".</p> <p>Expressions are not permissible as indices, e. g.: MyArray[v1+1]</p>

Key-word	Example	Description
ENUM access	<pre> Enumerator IF (i1 > 0) THEN v1 := forward; ELSIF (i1 < 0) THEN v1 := back; ELSE v1 := stop; END IF; </pre>	<p>Example:</p> <p>"Switch" is an ENUM type, "Forward", "Stop", and "Back" are the enumerators.</p> <p>The type of the variable v1 is "Switch".</p> <p>For more information, please refer to "ENUM TYPE Group, Page 146"</p>
Structure access	<pre> <STRUCTURE NAME>.ELEMENT o1.Temperature := i1;; o1.Speed := i2; </pre>	<p>The structure elements are separated with "." from the structure elements.</p> <p>Example:</p> <p>"o1" is a variable of type structure. "Temperature" and "Speed" are structure elements.</p> <p>For more information, please refer to "STRUCT TYPE Group, Page 148".</p>

7.4.2.5 Constants

Description	Example
Integer and bit strings (except BOOL)	-12 0 123_456 +986
Decimal representation	
Binary representation	2#1111_1111 (255 decimal) 2#1110 0000 (240 decimal)
Hexadecimal representation	16#FF or 16#ff 16#00F0 FFE0
Real	-12.0 0.0 0.4560 3.14159_26
Real with exponent	-1.34E-12 ODER -1.34e-12 1.0E+6 ODER 1.0e+6 1.234E6 ODER 1.234e6
BOOL	0 OR FALSE 1 OR TRUE
Time constants	Type identification with: „T#”, Time specification with: „d” (day), „h” (hour), „m” (minute), „s” (second) and „ms” (millisecond). T#12d12h17m42s T#16d 2h 5m
For all specifications, in general:	It is permitted to use a simple underscore for optical structuring.

7.4.2.6 Strings

Strings are enclosed in single quotation marks.

A \$ character followed by a hexadecimal number is interpreted as ASCII code.



Note

IEC also permits double quotation marks. These WSTRING have not been implemented at present.

Special characters allowed in strings

Combination	Interpretation when printing out
\$\$	Dollar character
\$'	Single quotation mark
\$L or \$l	Line feed (LF)
\$N or \$n	New line (NL)
\$P or \$p	Form feed (page)
\$R or \$r	Carriage return (CR)
\$T or \$t	Tabulator



Note

A \$R\$L (Carriage Return / Line Feed) is equivalent to a \$N (New Line) and is, hence, displayed automatically in the function block during further processing as \$N (see entry in STANDARD VALUE and display under VALUE).

Index	Data type	Name	Default	Description
Variable type: Input				
1	STRING	i1	'\$R\$L'	'\$N'
Variable type: Output				
1	INT	o1	0	

figure 71: Variable entry

Characteristics and examples of strings

Example	Explanation
"	Blank string (Length = 0)
'A'	String of length one, contains the character A
' '	String of length one, contains the blank character
'\$'	String of length one, contains the single quotation mark
'\"'	String of length one, contains the double quotation mark
'\$R\$L'	String of length two, contains the ASCII characters for CR and LF
'\$\$1.00'	String of length five, contains "\$1.00"
'ÄË'	String of length two, contains "Ä" and "Ë"; In one case, directly as ASCII characters and
'\$C4\$CB'	in the second case, with the corresponding hexadecimal code of the extended character table (see "Character tables, Page 322")

7.4.3 Macro block

You use macros in order to combine associated functions and thus, achieve a clear program layout.

Properties:

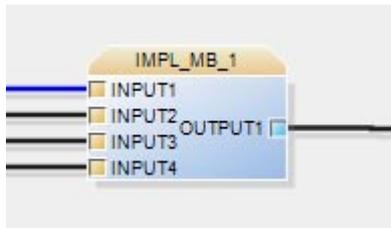
- You can export macros.
- You can copy macros to the global folder, and, as a result, you can use them several times and even in other projects.
- Macros may contain other macros and, of course, user-created function blocks, too.
- No OTCs, switches and sliders are permissible within macros, but IPCs are allowed. Links to other program components are permissible only with input and output connectors.
- You cannot use any hardware input and output resources directly within macros.
- A macro that has been created can be expanded again, i. e. the macro is resolved and the blocks that it contains are displayed at the next higher level.

7.4.3.1 Creating a Macro Block

Macros are created manually in the same way that function blocks are created.

Procedure

1. Position the mouse pointer on a free location in the program window and call up "New... - New Macro Block..." in the context menu. A dialog box is displayed for entering the block names and variables. For more information, please see "Function Blocks, Page 123".
2. Exit the dialog with <OK>. A blank macro block is displayed.



3. Double click on the macro block displayed in order to open the internal graphical programming interface.
4. Place and manage the function blocks or other macro blocks within this macro block so that you achieve the desired functionality.

Example

In this example the integer input is monitored for changes and every change is counted and placed at the output.

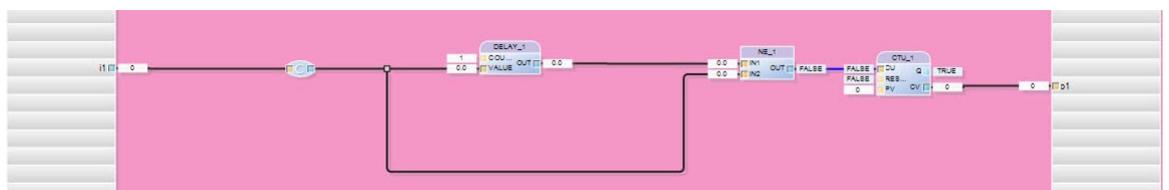


figure 72: Integer monitoring

As in the case of every program, a new register is created for the contents of the macro with the macro name within the program designer.

Please note the evaluation context here. You come to the calling level by clicking on this. For further information, see "Arrangement of the Tabs and Programming Windows, Page 62".

7.4.3.2 Opening a Macro

Procedure

- Double click on the macro block instance in a program or macro in the Workspace Explorer.

7.4.3.3 Combining existing components into a Macro Block

ibaLogic provides the option of combining several blocks already existing into one Macro Block.

- To do this, select the blocks that need to be combined, as illustrated in the following diagram.

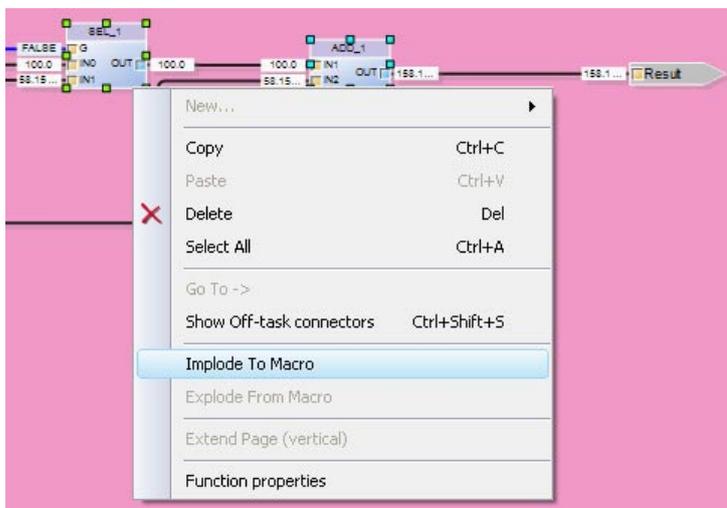


Note

Please note that,

- you also mark the associated links when making the selection.
- no OTCs have been marked.
- links whose target or source block are not selected, are created as macro input or output.

- Open the context menu using one of the elements selected.
- Select the option "Implode To Macro".
The "Edit Function Block" dialog box is displayed.



- Assign a meaningful name to the new macro block and to the inputs and outputs. Assign meaningful names that conform to the IEC standard.



Tip

You can also make these changes subsequently by clicking on the macro created using the right mouse button and selecting the macro properties.

Result

The result is a new macro block (IMPL_MB_1) having the same functionality as the blocks selected previously.

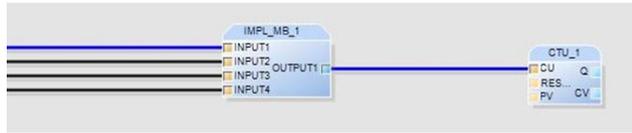


figure 73: Macro block (IMPL_MB_1)

- You can open the macro by double clicking and continue to edit the graphical elements.

In the online mode, you can also see the current values in the value pads depending on the evaluation context.

For further information on the evaluation context, please see "Arrangement of the Tabs and Programming Windows, Page 62".

7.4.3.4 Expanding a Macro Block

An existing macro block can be expanded again. In doing so, the blocks defined are placed at the next higher level.

Procedure

1. To do this, mark the macro block.
2. Select "Expand From Macro" in the context menu.

Example

A simple macro block having an internal adder that adds both inputs needs to be expanded again.

Result

The following diagram appears after expanding:

- The adder has been revealed.
- The original macro definition, however, continues to be available in the block library.

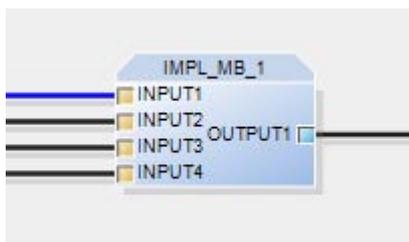


figure 74: Macro block MB_Set-point_1

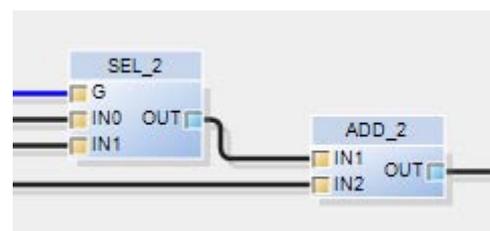


figure 75: Expanded connection

7.4.4 Creating your own DLLs

Creating macros and function blocks using ST is a very easy option for handling several tasks in the field of automation technology. But just as it is easy to create the macros and function blocks, it is also simple to copy them and to understand their contents, i. e. their function.

However, sometimes it is desirable to disclose less of one's own technological competence and, instead, for example, in case of a highly intelligent process-oriented technical solution, it is desired to prevent further uncontrolled proliferation of this technological know-how.

In such a case, it is beneficial to have the option of creating your own DLLs that contain the knowledge only in compiled form and, thus, cannot be extracted easily.

You can also realize these special connections

- in order to create complex blocks.
- in order to work with the Windows environment.
- in order to allow tasks to execute in your own thread, among others.

In this manner, you can also integrate another high-level language under certain conditions.

You can then see the DLL created in ibaLogic as a completely normal function block with the name, inputs and outputs. This can be differentiated from an ST function block only in the fact that you do not see any code in the program component.



Other documentation

This section contains only a brief overview, and detailed instructions are furnished on the CD supplied (e. g. Manual "ibaLogic_DLL-Erstellung_in_C++_v2.0_de.pdf" in the \ibaLogic_V4.x.x\Samples-DLL\ directory).



Note

Using DLLs requires a license. The DLLs are not evaluated without a valid dongle.

Compiler

All DLLs written in C++ or Fortran are supported.

The following compilers have been tested for writing and compiling the DLLs:

- Microsoft Visual C++ 5.0
- Microsoft Visual C++ 6.0
- Intel Visual Fortran 10.0
- Microsoft Visual C++ 2005, 2008, 2010

However, there are still differences for the two device classes with ibaLogic (WinXP or PADU-S-IT). DLLs for the PADU-S-IT platform must be compiled specifically for Windows-CE.

7.4.4.1 Source Files and Descriptions Required

The following source files and descriptions, which are supplied on the ibaLogic installation CD, are necessary for creating a DLL:

Descriptions:

- Manual on creating a DLL with C++
(for WinXP and PADU-S-IT)
- Manual on creating a DLL with Fortran
(only available for Windows XP)

Files: (Please refer to the descriptions for the exact names of the files)

- Framework file:
It contains the procedures and the DLL body; the user can add inputs or outputs or modify the procedures, InitEvaluation, Evaluate and ExitEvaluation. Either in C++ or Fortran.
- Other files depending on the language:
Assignment of DLL procedures and numbers, interface definition etc. It is not necessary for the user to make any modifications here.

7.4.4.2 Requirements and Notes

You should take note the following when creating DLLs:

- The runtime of the DLL increases the runtime of the tasks in which they are called.
- iba recommends that you remove time-consuming functions to threads.
- ibaLogic can be started as the executing program to test the DLL.



Important Note

ibaLogic cannot detect and trap programming error in a DLL, which means that such errors can lead to ibaLogic "crashing". Please bear this in mind as a user when creating a DLL.

7.4.4.3 Integrating the DLL into ibaLogic

When the DLL was created, it must be copied to a folder of ibaLogic. (Usually "C:\Program Files\iba\ibaLogic v4\Server\DII").

After ibaLogic Server restarts the next time, this DLL is available as a function block in the "CUSTOM" folder and can be dragged & dropped like any other block in a program and integrated in it.

Example

The "Para_File_Read_Store_DII" has been created and copied to the folder. It is contained in the "CUSTOM" group.

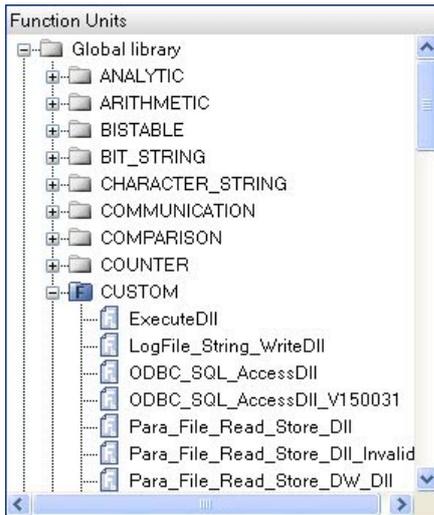


figure 76: Para_File_Read_Store_Dll in the function block navigator



figure 77: Para_File_Read_Store_Dll as a function block in the program

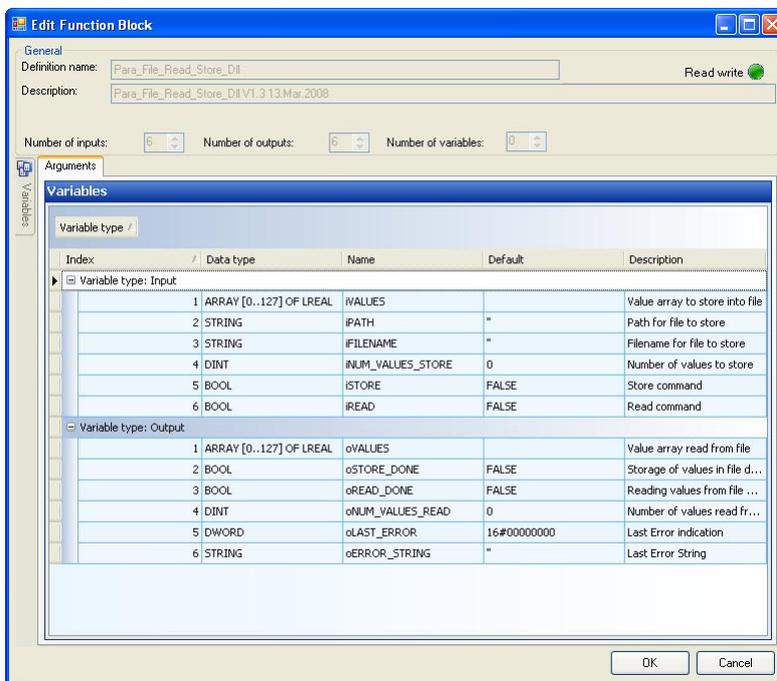


figure 78: Para_File_Read_Store_Dll properties

- ➡ The "Edit Function Block" window is displayed when you double click on the function block.
You cannot see the code. The inputs and outputs including their descriptions are visible.

7.5 Data types

A data type is assigned to each variable.

In contrast to version V3, ibaLogic-V4 not only supports the elementary data types and arrays, but also composite (structures) and other user-defined data types.

The data types that can be used in ibaLogic can be divided into the following categories:

- Standard data types
- Composite data types such as ARRAY, STRUCT and ENUM,
- Derived data types that are formed from the groups mentioned above.

As a user, you can define your own specific data types of the "composite" or "derived" category.



Note

For more information, please refer to "Data types, Page 278".

7.5.1 Define Data Type

- ➔ Click on the "Data Types" button.

The folder is displayed in the navigation area as a tree.

The following folders are created for the **non-elementary** data types in a global library and also under each project of the workgroup:

- Direct derived types**
Standard data type with a fixed default value
- Sub-range types**
Standard data type with a fixed default value and limited range of values
- String derived types**
String data type with a fixed length and default text.
- Enum types**
Enumerations: Names are defined instead of integer values
- Array types**
Array of elementary data types with a fixed dimension and depth.
- Struct types**
Structure consisting of elementary data types

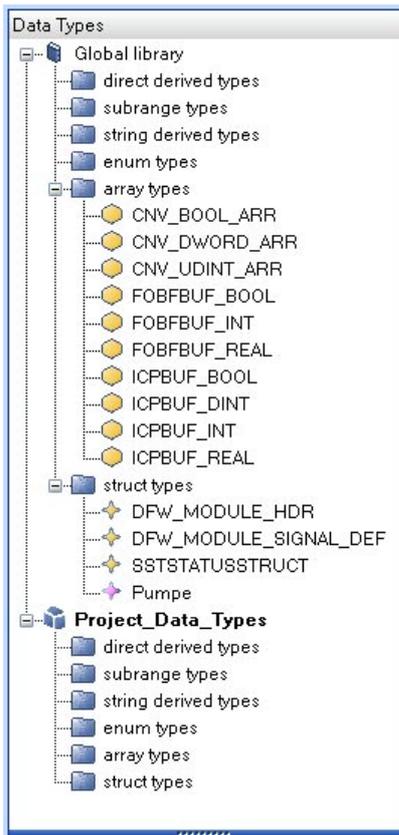


figure 79: Data Types

The "Array types" and "Struct types" contain data types that have already been predefined by ibaLogic.

These are:

- ❑ **CNV_BOOL_ARR / CNV_DWORD_ARR / CNV_UDINT_ARR**
Single-dimensional arrays with 58 elements. These are required when importing former ibaLogic V3 projects.
- ❑ **FOBFBUF_BOOL / _INT / _DINT / _REAL**
Single-dimensional arrays with 256 elements, usage in "Buffered Mode". For more information, please see section "Buffered Mode, Page 192".
- ❑ **ICPBUF_BOOL / _INT / _REAL**
Single-dimensional arrays with 1,024 elements, used for connecting analog inputs in the PADU-S-IT platform.
For more information, please see "ibaPADU-S-IT Platform, Page 204".
- ❑ **DFW_MODULE_HDR / DFW_MODULE_SIGNAL_DEF**
Structure for transferring data to the DAT_FILE_WRITE block.
- ❑ **SSTSTATUSSTRUCT**
Structure for coupling the diagnostics information to the Profibus master card SST.



Note

You can suppress the display of the data types that are predefined and generated automatically if you select the "Tools – Options – General – System" menu and enable the "Suppress Generated Data Types" option.

You can use the data types in the program even if the display is suppressed.

Procedure

You can define a data type in different ways:

- Under the project
- In the global library
- During the creation of a function block

7.5.1.1 Under the project

1. Click in the function tree with the right mouse button on the desired category under the project.
2. Select "New" in the context menu.

7.5.1.2 In the global library

1. Click on the function tree with the right mouse button in the global library on the desired category.
2. Select "New" in the context menu.

7.5.1.3 When creating a Function Block

The option of creating a new data type is provided in the selection box for the data type of a variable.

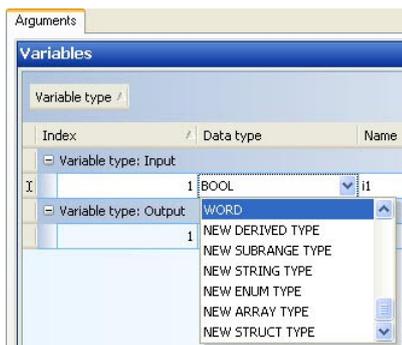


figure 80: Creating a data type for a block

Procedure

1. Create a variable with the appropriate data type.
2. Test the data type created to ensure that it is error-free. If the test was successful, confirm the entry with <OK>.

Result

If the syntax is error-free, the data type is created under the category selected.

7.5.2 Modify Data Type



Note

A data type that is already in use cannot be modified.

When you exit the dialog with <OK> or <Accept>, your attention is drawn to the fact that you can create a copy under a different name.

Procedure

1. Click with the right mouse button on the data type.
2. Select "Properties" in the context menu.
3. Modify the parameters.

7.5.3 Delete Data Type

Requirement

You are not using the data type to be deleted.

Procedure

1. Click with the right mouse button on the data type.
2. Select "Remove" in the context menu or press the function key.

7.5.4 Manage Data Type

Copy to the global library

If you need to use a data type, which is defined in one project, in another workspace also, the data type must be copied to the global library.



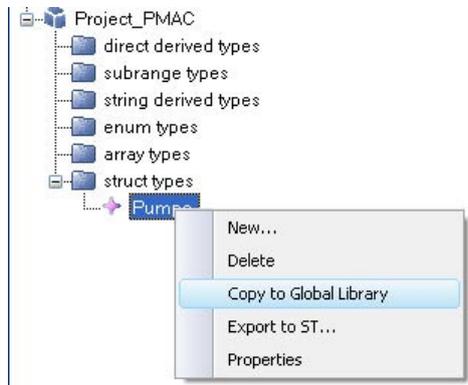
Note

If you use data types from the global library, they are automatically copied to the project.

If you use a data type from another project, this has to be copied to the global library first.

Procedure

1. Click with the right mouse button on the data type under the project.
2. Select "Copy To Global Library" in the context menu.



Note

If you copied an array to the global library and then change the original, you have two arrays with the same name, however, with different contents. When selecting in the FB, [GLB] is put in front of the array from the global library.

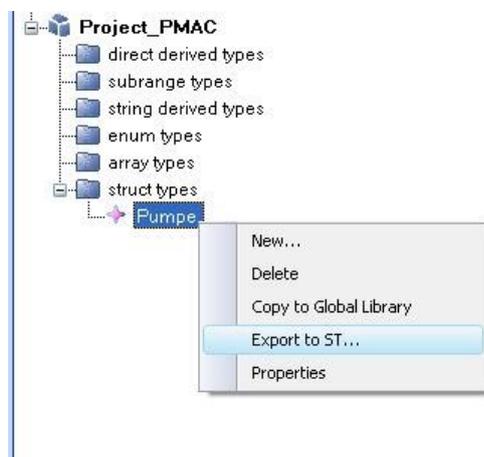
7.5.5 Export Data Type

If a data type, which is defined in another database under the global library or in a project, also needs to be used in another database or in another programming tool, the data type must be exported as a text file.

Procedure

1. Click with the right mouse button on the data type.
2. Select "Export to ST" in the context menu.

The "Export" dialog box is displayed.



3. Specify the target folder and file name.

7.5.6 Import Data Type

Requirement

ibaLogic is not in the online mode.

Procedure

- Click on the "File – Import – Structured Text" menu.

7.5.7 Use Data Type

After a data type has been defined you can use it:

- during the creation of a function block
- during the creation of a structure and/or array data type
- when creating inputs and outputs

7.5.7.1 During the Creation of a Function Block

- Select the user-defined data type under the "Data Type" column while creating a variable in the block editor.

7.5.7.2 During the Creation of a Structure Data Type

- Select a user-defined data type in the "Data Type" selection box while creating the structure elements in the data type editor.



Note

You cannot directly access data types which you defined in a project of another workspace.

For doing so, use the Export/Import function or the global library.

7.5.8 User-defined Data Types

Procedure

- Click with the right mouse button on a data type group.

The "Edit data types" dialog box is displayed.



For all data types, this dialog box consists of:

- "General" section (identical for all data types)
- "Type Properties" section
- "Elements" section

General

- Name:
Name for the user-defined data type. The data type is then available under this name in the selection boxes.
- Description:
Any text for the description of this type. The description can be seen only here in the definition.

Type properties

- Type:
Defines the data type.
- Default value:
Initial value (Preset value)

7.5.8.1 DIRECT DERIVED TYPE Group

This is to define an elementary data type for which a new name and a default value can be specified.

With this, for example, constants such as the number "Pi" can be defined with the LREAL data type.

7.5.8.2 SUBRANGE TYPE Group

This is an integer data type with a limited range of values and a default value.

You can use it, for example, to define indices for arrays having a specific depth.



Important Note

This data type is **not** limited. There is merely an error message at runtime in case the range is exceeded.

This data type is only checked in case of direct assignments during compilation, there is no runtime checking as to whether the range is exceeded.

7.5.8.3 STRING DERIVED TYPE Group

This is a string data type having a limited length.

You can use it to define text strings having a fixed initial value, for example, for error messages.



Note

The maximum length of a string is 250 characters.

7.5.8.4 ENUM TYPE Group

A data type of the category ENUM TYPE is an enumerator.

The data type is used to designate the values of a variable symbolically, e. g. the position of a switch.

Example: Data type "Switch"

You would like to create a data type "Switch", which has the 3 positions, FORWARD, STOP and BACK.

To do this, you need to define the ENUM TYPE Switch, with number 3 and the switch positions as enumerators.

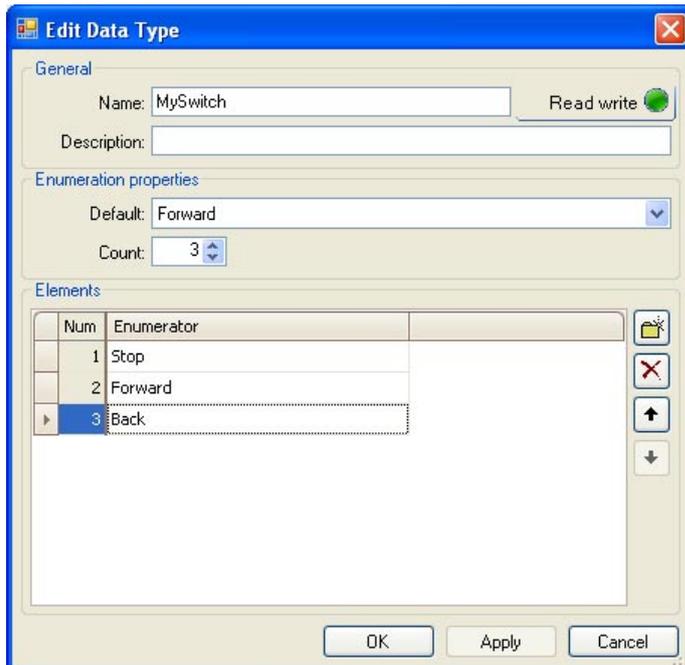


figure 81: "Edit data types" dialog box

Please note that you access the individual enumerator values using "Enumerator" in "Structured Text".

Assign and retrieve values:

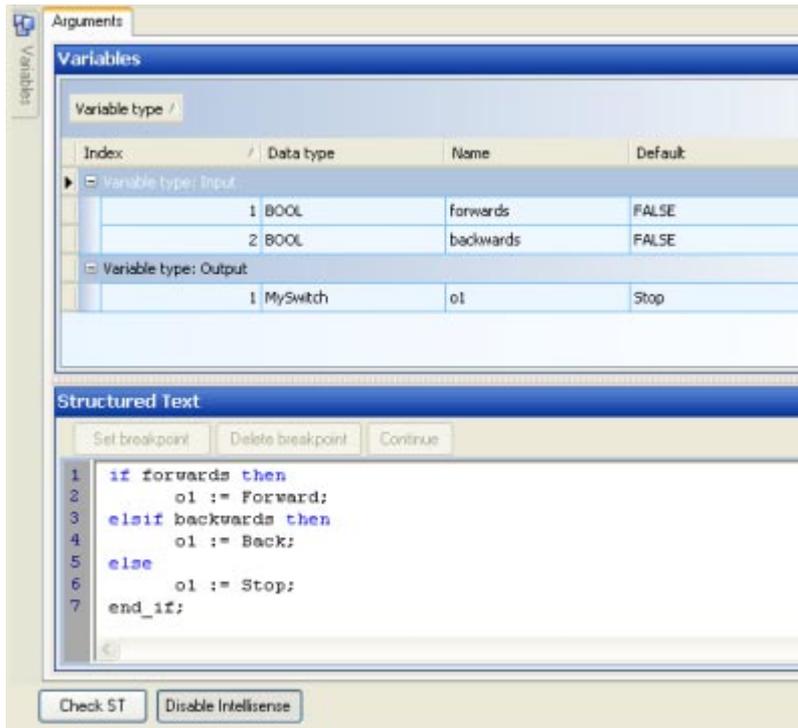


figure 82: Variables Editor 1

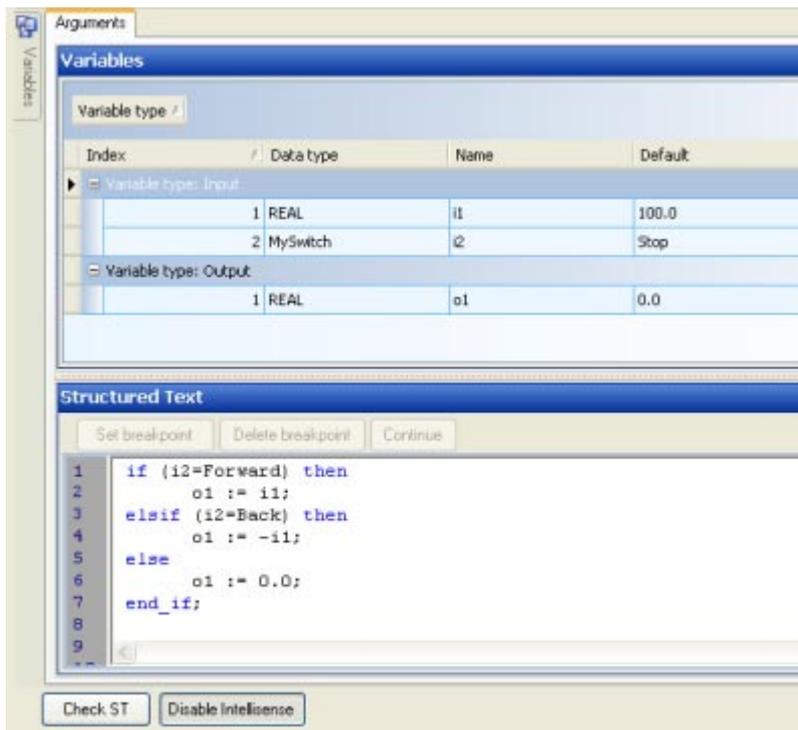


figure 83: Variables Editor 2



Note

Please note that you cannot assign integer values to the enumerators.

Exception:

An OPC connector is declared as type Enum and read or written externally. In this case, the OPC Client writes or reads the enumerator number as an integer.

7.5.8.5 ARRAY TYPE Group

Arrays are single-dimensional or multi-dimensional fields. All elements of an array have the same data type. However, this is not restricted to the elementary data types, but you can also form arrays of user-defined data types, structures, strings or arrays.

Example: 2-dimensional integer array

Parameter	Explanation
Type	Base type of the array elements
Number	Number of dimensions
Default value	Default values of the array Examples 1-dim-Array: [1.0,2.0,3.0] 2-dim-Array: [[1.0,2.0,3.0],[4.0,5.0,6.0]]
Lower / Upper limit	The value range of the element index determines the depth of the individual dimensions. Maximum value: 0 to 32,766

Access to the elements of an array in Structured Text:

i1 is a variable of array type. *o1*, *o2*... are variables of element type;

- 1-dim- Array: `o1 := i1[0];`
- 2-dim- Array: `o1 := i1[0.0];` (* 1st element of the 1st dim *)
 `o2 := i1[0,1];` (* 2nd element of the 1st dim *)
- array_of_array: `o1 := i1[0][0];` (* 1st elem. of the 1st array *)
 `o2 := i1[0][1];` (* 2nd elem. of the 1st array *)
 `o3 := i1[1][0];` (* 1st element of the 2nd array *)
- array_of_struct: `o1 := i1[0];` (* 1st structure of the array *)
 `o2 :=` (* elem. of the 1st structure *)
 `i1[0].elem;`



Note on Structured Text

The indices of arrays can be only variables having "Int" data type. In contrast to ibaLogic V3, **no** expressions such as [ix+4] are allowed.

7.5.8.6 STRUCT TYPE Group

In contrast to arrays, you can group variables having different data types under a structure. You have to define the elements separately, and while doing so, you can use all data types defined so far, including the user-defined data types and arrays.

You can define a name, description and default value for each element of the structure.

Example: Pump

You need a "Pump" data type for the pump "Type E7F99" with the properties "temperature", "speed", "state" and "error".

For this purpose, you define the "Pump" data type with the description "Pump Type E7F99" and the number 4. "Elements" defines the associated properties.

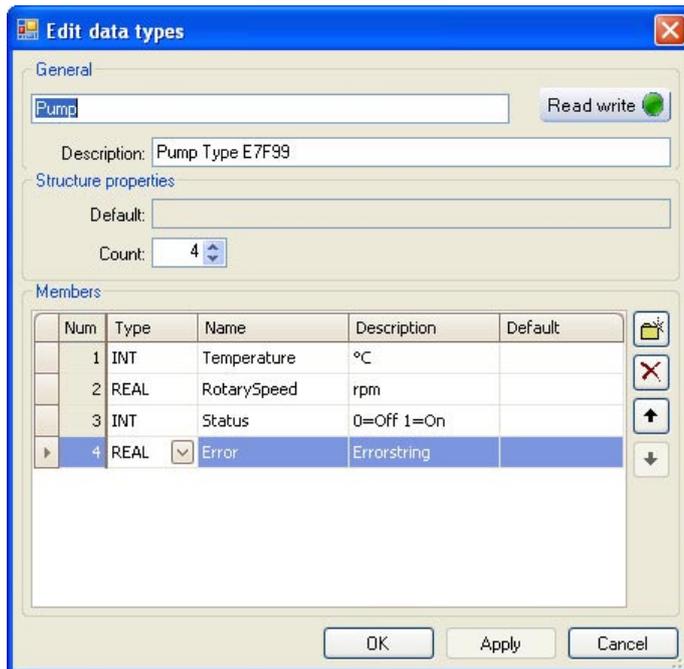


figure 84: "Edit data types" dialog box

Parameter	Explanation
Type	Base type (even user-defined data types are allowed).
Name	Name of the element.
Description	Personal explanation of the data type.
you select the settings	Initialization to a default value (Preset).

Access to the elements of the structure in Structured Text:

```
1 (*o1 is a variable of structure type pump,   i1, i2... are
   variables of element type; *)
2
3
4     o1.Temperature := i1;;           (* of INT data type *)
5     o1.Speed := i2;                 (* of REAL data type *)
6
7 (*v1 is a variable of Pump structure type;*)
8
9 if      (v1.Temperature > 80 ) then
10     v1.Status:= 99;
11     v1.Error:= 'Temp. too high';
12 else
13     v1.Status:= 0;
14     v1.Error := 'No error';
15 end_if;
```

figure 85: Structure in Structured Text

8 Program Elements

A graphical ibaLogic program contains the following elements:

- Blocks
- Inputs and Outputs
- Links (Connections)
- Converters, splitters or joiners added automatically
- Comments

8.1 Create Program Element

You can create all elements that a graphical ibaLogic program can contain.

Procedure

1. Open the context menu by clicking with the right mouse button on a free area in the programming field.
2. Select "New..." in the context menu.
3. Select the desired program element.

8.2 Mark Program Elements

You can mark individual or multiple program elements in the following manner.

Procedure

1. Select the desired element by clicking with the left mouse button (single selection).
2. Select the desired elements by clicking with the left mouse button and pressing the <Shift> or <Ctrl> simultaneously (multiple selection).
3. Drag a rectangle (Lasso) over one or more elements to be selected by clicking on the left mouse button.
If doing so, existing connection lines and converters, if any, between the blocks are also marked.
4. Select all elements by pressing the <Ctrl> + <A> keys.

Result

The elements marked are displayed with green, blue or gray dots.

When you mark several elements, one element is always green. This is the reference point of the grouping.

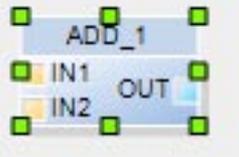
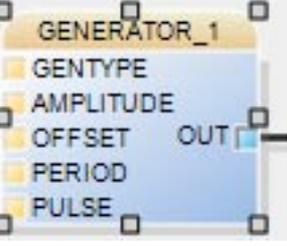
	One element is marked.
	One element is marked, but the focus lies on another element.
	Many elements are marked. The element marked green is always the main element of the group marked.

figure 86: Selected element

8.3 Move Program Element

You can move the blocks (and connections) already marked by keeping the left mouse button pressed.

Procedure

- ➔ Move one or more elements selected by keeping the left mouse button pressed.

Remarks

This applies to lines with limitations.

8.4 Align Program Elements along an Edge

All program elements can be aligned along an edge. You can use this for obtaining a clear block layout.

Procedure

1. Mark the elements to be aligned (Blocks, Intra-page connectors and Off-task connectors).
2. Select the desired function the "Function Diagram - Align".

Remarks

This is not applicable to inputs / outputs and lines.

8.5 Copy Program Element

You can copy individual or multiple program elements.

Procedure

1. Mark the elements to be copied (Blocks, Intra-page connectors and Off-task connectors).
2. Press the key combination <Ctrl> + <C> to copy the elements selected to the clipboard.
3. Press the key combination <Ctrl> + <V> to copy the elements from the clipboard into the programming field.



Tip

Instead of the key combination, you can also select "Copy" and "Add" in the context menu.

If you have selected multiple blocks, the connecting lines between these blocks also get copied.

This is not applicable to inputs and outputs and lines marked individually.

8.6 Delete Program Element

You can remove individual or multiple program elements.

Procedure

1. Mark the elements to be deleted (Blocks, Intra-page connectors and Off-task connectors).
2. Press the key.

Remarks

Instead of the key, you can also select "Remove" in the context menu.

8.7 Generate Input / Output Variables

Prerequisite

You have selected the "Inputs - Outputs" button.

Procedure

- Drag an input or output variable at any position in the left or right input or output border.



Note

In a **program**, an input and output **can be created** only once.

In a **project**, an **input can be used** several times. An **output** can be used only **once**.

8.8 Graphical Connections

In graphical programming, a graphical connection is used to transfer the results of one function to another.

There are 3 different forms for this:

- Direct connectors
- Intra-page connectors
- Off-task connectors

8.8.1 Direct Connectors

8.8.1.1 Types of connection lines

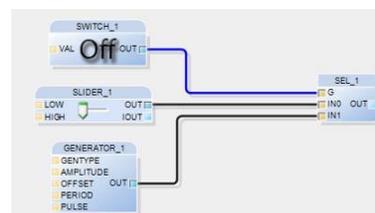
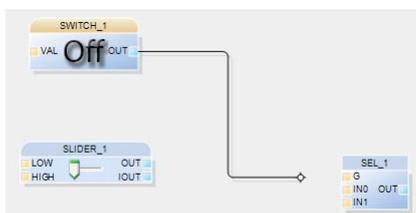
ibaLogic uses line types of different colors that represent different groups of data types.

Line type	Explanation
	Binary connectors are displayed according to their status, red (TRUE) or blue (FALSE).
	Arrays are displayed using green lines. Only arrays having identical length and data type can be connected with one another.
	Structures are displayed in orange color.
	Enum types are displayed in yellow color.
	All other elementary data types are marked with black connectors (e.g. INT, REAL...)

8.8.1.2 Create Direct Connector

Procedure

1. Click with the mouse on the output connector of a block.
2. Keep the left mouse button pressed and drag a connector to the input connector of a block.



Remark

If the result of one block is used in multiple blocks, generate a branch by dragging a line from one input connector to another existing one.

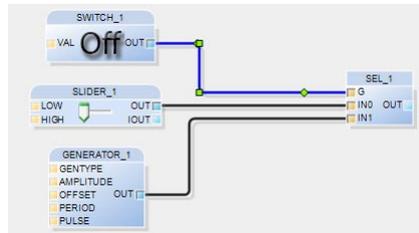
Near a connectable connector or connectable line, the mouse cursor jumps to the connector or the connecting line (Magnetic effect).

8.8.1.3 Modify Direct Connectors

Procedure

1. Mark the connector that you wish to modify.

The marking is displayed by small green squares and diamonds.



2. Modify the line by moving the green squares with the mouse.
3. Click with the mouse on the green diamond on the connecting line in order to wire the line connection.
4. Drag the end to a blank area, which deletes the connecting line.
5. Drag the end to another connector. The connecting line is reconnected.



Note

Connecting lines that you have arranged manually are re-evaluated by the auto-router when the associated block is moved. Your modifications are rejected as a result.

8.8.2 Intra-Page Connectors

An intra-page connector (IPC) merely represents a drawing simplification. In the process, the IPC replaces a connecting line.

This is recommended when several objects on one page need to be connected or "long" connections are required across multiple pages. The IPC is not a programming object, but merely acts as a line substitute.

The IPC can - be used as direct connectors - only within a program or macro level. You cannot have connections from a macro to the call level. You must define inputs and outputs in the macro block for this purpose.

8.8.2.1 Create Intra-Page Connectors

Create Intra-Page connectors as line substitutes.

Prerequisite

You can generate an IPC at an input connector only if an "IPC Source" has been defined earlier.

Procedure

- ➡ Press the <Ctrl> button and simultaneously drag a connecting line from one output connector to a free location in the programming field.

- Menu procedure similar to creating off-task connectors. However, major modification.
 - Create IPC source
 - Connect IPC (as described here)
 - Connect IPC via menu

Remark

To connect an IPC to an input, follow the same procedure.

Hold down the <Ctrl> key and drag the line from an input connector to a free area in the program field. Subsequently, the "Existing IPCs" dialog opens. Select the corresponding IPC and quit the dialog by clicking <OK>.

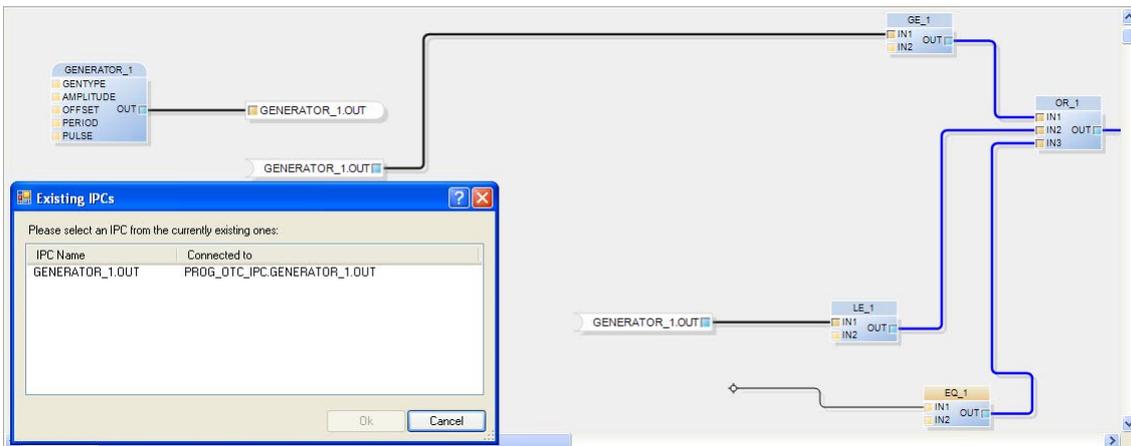


figure 87: Properties window

8.8.2.2 Modify IPC Names

ibaLogic creates a name automatically, consisting of "Block instance name.connector name". This name can be changed.

Procedure

1. Double click on the IPC source.
The "Edit IPC " dialog box is displayed.
2. Assign a name and a comment to the IPC source. Assign meaningful names.

Result

The modification is accepted automatically in all "IPC Targets" connected. "IPC Targets" cannot be directly modified.

8.8.2.3 Track IPC

Load the corresponding program page of the IPC selected.

Procedure

1. Click with the right mouse button on an IPC.

2. Select "Go To ->" in the context menu.
You can then see the generator (output connected and all consumers (inputs) connected).
3. Click on any connection displayed.

Result

The appropriate program page is loaded and the IPC is marked.

Context menu	Explanation
01. -> Generator_1.OUT	Generator
02. °Generator_1.OUT ->	Marked IPC
03. Generator_1.OUT ->	Consumer

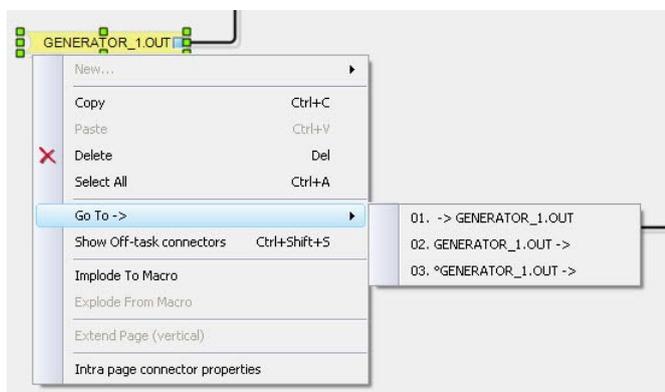


figure 88: Track IPCs

8.8.3 Off-Task Connectors

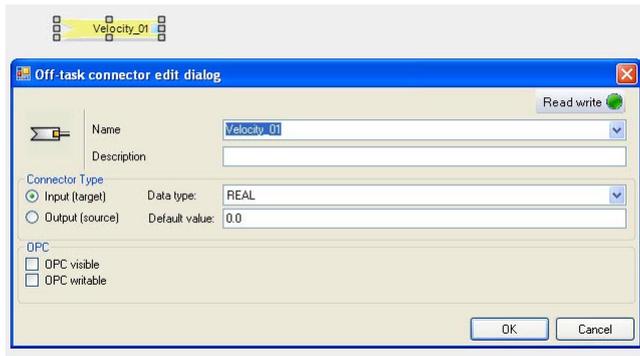
Off-task connectors (OTC) are used as program-independent connecting elements are always required when there is communication between programs.

In addition, OTCs can be set as read-enabled and write-enabled for OPC Clients.

8.8.3.1 Create Off-Task Connectors

Procedure

1. Position the mouse pointer at a free location in the programming field.
2. Open the context menu by clicking on the right mouse button.
3. Select "New... - New Off-Task Connector".
The "Edit Off-task connector" dialog box is displayed.



4. Assign the parameters required.



Note

The OTC name must conform to the IEC naming convention. See "Naming conventions, Page 277".

Establish a program-independent connection

Procedure

Method 1:

1. First generate the output OTC (Source) by filling up the dialog box.
2. Copy the output OTC.
3. Add the output OTC in the target program.
By doing so, the parameters are accepted but the direction is reversed.

Method 2:

1. Create an OTC in the target program.
2. Select the name of the associated output OTC from the selection box.
In doing so the other parameters get accepted.
3. You must set the direction to "Input".

OPC Properties

You can specify the following OPC properties to the OPC.

Selection boxes OPC Properties	Explanation
OPC visible	Specifies whether this connector is visible in the OPC name space.
OPC write-enabled	Specifies whether an OPC Client should write to this connector.

Further information, please refer to "Setting the OPC Variable Parameters, Page 208".

Rules for creating OTCs

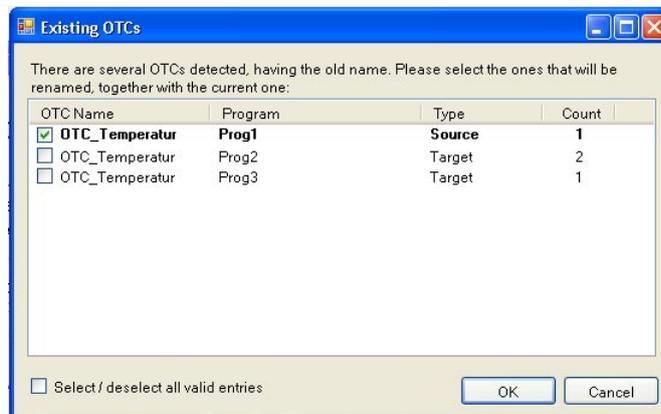
- An output OTC must be unique in the project.
- Multiple input OTCs can be created for an output OTC. You can do this even within a program, but not in the program in which the output OTC is placed.
- An input OTC can have only one data source, either OPC enabled or an associated output OTC.
- If you create an output OTC with the name of the input OTC, this input OTC is no longer write-enabled for the OPC.
- An input OTC that does not have any data source can be used as a constant / parameter.

8.8.3.2 Rename OTC

Procedure

1. Select the OTC that is to be renamed.
2. Open the OTC properties by means of the context menu or by double-clicking on OTC.
3. Change the name of the OTC and leave the properties dialog by clicking <OK>. If the OTC already has connected targets, the "Existing OTCs" dialog box is displayed.

In this screen, it can be determined whether all or individual connected OTCs are to be renamed. In this manner, for example, you can assign the right name to an OTC having an incorrect name by correcting the name.



After quitting the dialog by clicking <OK>, all selected OTCs are renamed.

Remark

If a target is present several times in a program, the number can be seen from the COUNT column.

8.8.3.3 Track OTCs

Procedure

1. Click with the right mouse button on an OTC.
2. Select "Go To ->" in the context menu.
You can then see the programs in which the OTC is generated and used.
3. Click on any connection displayed.

Result

The appropriate program page is loaded and the OTC is marked.

Context menu	Explanation
01. -> OTC_Temperature: Prog1	Generator
02. OTC_Temperature -> : Prog3	Consumer

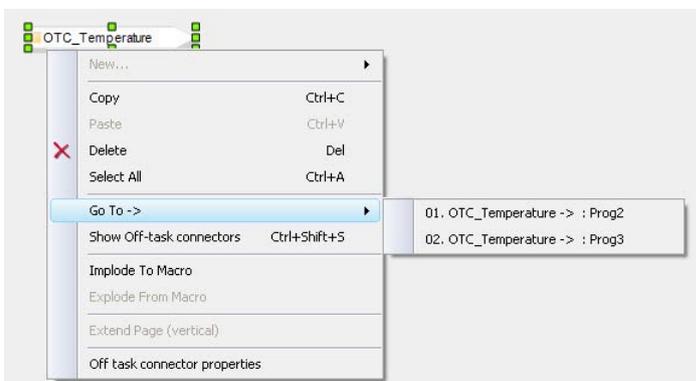


figure 89: Track OTCs

8.8.3.4 List of all OTCs

Display of all defined OTCs in the project.

Procedure

1. Position the mouse pointer at a free location in the programming field.
2. Open the context menu.
3. Select "Display Off-Task Connectors".
 The dialog box that contains the list of all OTCs defined and sorted in alphabetical order is displayed.
 Navigate within the list by entering the starting alphabet or by double clicking on an OTC.
 This displays a list of all OTCs defined in the project.

Remark

You can also call this function via the "Function diagram - Display Off-Task Connectors" menu option or the key combination <Ctrl>+<Shift>+<S>.

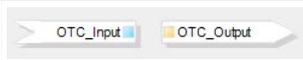
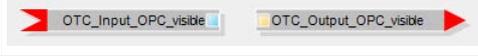
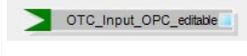


Note

You can leave the dialog box open and continue working with the program. The list is updated automatically when OTCs are created or removed. The dialog box can be positioned wherever desired and can also be docked to the border of the programming window.

8.8.3.5 Display

The various properties are identified in color for better orientation:

Display of the OPCs	Description
	Displays inputs and outputs that are connected
	Displays inputs and outputs that are visible to the OPC, write-enabled and read-enabled
	Displays an input that is read-enabled and write-enabled for the OPC
	Displays inputs and outputs that are not connected

8.9 Converters, splitters, joiners

8.9.1 Converter

Automatic Addition of the Data Type Conversion

In conventional CFC editors you can connect interfaces that are not of the same data type. ibaLogic adds a converter automatically if a meaningful conversion is possible.



Note

To use detailed converters directly displaying the conversion, disable the "Iconic Display of the Converter" function in the options.

see: "Tools" - "Options" - [Editors] - [Diagram]

This automatic converter is displayed with reduced size in order to save space in the programming field.

When you go over it with the mouse pointer, it displays the conversion hidden below it as a tooltip.

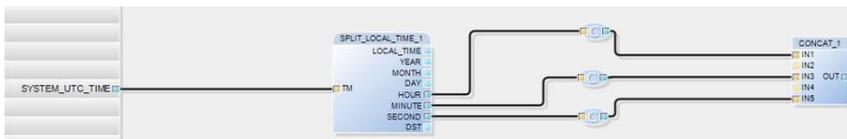


figure 90: Data type conversion



Note

Blocks having non-typed connections get a data type only when a connector is put in place. This data type is then accepted for all connections and remains when you remove the last connection again.

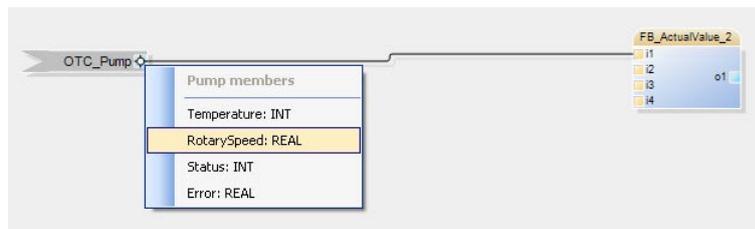
If you wish to connect two non-typed connections a dialog box opens in which you can select a permissible data type.

8.9.2 Splitter

You would like to remove elements from a data structure for other evaluations. Using conventional methods, you have to create a user-defined block in which you use structured text to assign the structure of the output connectors to individual elements. IbaLogic automatically creates this block, known as splitter, for you.

Procedure

1. Drag a connecting line between one block input connector to a structure output connector of a block or an OTC input. A selection box pops up.
2. Select a structure element.



Remarks

In this manner, you can access other structure elements.

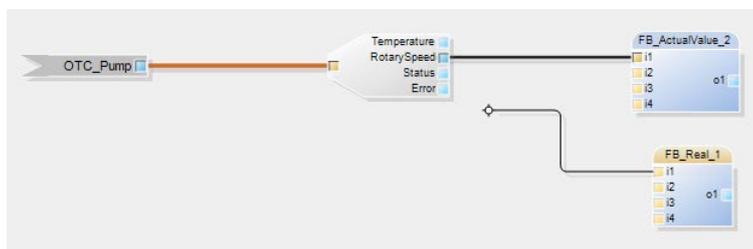


figure 91: Splitter

8.9.3 Joiner

If you try to connect an output connector with the input connector of a structure, IbaLogic provides a menu where you can select one of the structure elements. Based on this, IbaLogic adds a joiner block to whose inputs you can connect other signals.

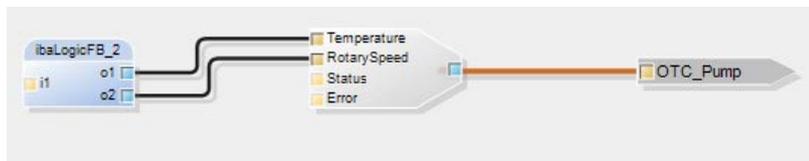


figure 92: Joiner

8.10 Comments

Comments are graphical elements that you can add at any free location in the programming field. You can cover connectors. These are visible through the transparent comments field.

The comments field has a pointer that can be docked to the function to be described.



Procedure

1. Position the mouse pointer at a free location in the programming field.
2. Open the context menu.
3. Select "New... - New Comment".

9 PMAC Runtime System

9.1 Overview of Online and Offline Modes

The following menus or icons are available in the menu for operating the runtime system:

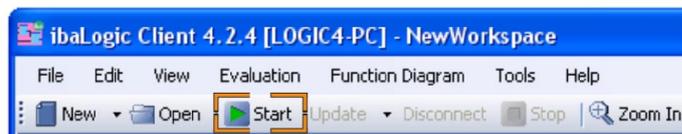
- Start (Menu "Evaluation" and button in the toolbar)
- Stop (Menu "Evaluation" and button in the toolbar)
- Store project on target ("Evaluation" menu)
- Delete stored project from target ("Evaluation" menu)
- Disconnect (Button in the toolbar)
- Update (Button in the toolbar)

9.2 Start Runtime System

In contrast to conventional automation systems, the steps of "Compilation" and "Loading" take place automatically in the background.

Procedure

1. Click on the <Start> button in the toolbar.



2. Confirm the "Start Evaluation..." dialog with "Yes".



Note

This request can be disabled in the ibaLogic options to start the evaluation in the development and test environment by pressing the <Start> button or F5.

To disable the query, open the options with "Tools"->"Options" and enable the "Program: Start evaluation" option under [General]->[Messages]->[Confirmations].

Result

The following actions are performed:

- The project is compiled.
- The project is transferred to the PMAC.
- Program execution commences.
- The evaluation time is displayed in the program window toolbar.
- All value pads are displayed.

- The value pads in the visible region are provided with current values.
- The background color of the programming field in the client changes to pink.
- The program is now in online mode.

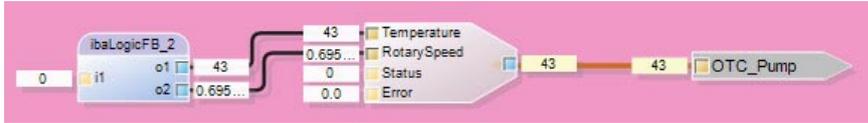


figure 93: Online mode

Errors during compilation, loading, etc. are displayed in the event window. By default, this is located below the programming field. It can, however, be concealed or placed at any position desired and docked there.



Tip

A special highlight of ibaLogic is the fact that you can (almost) carry out the entire programming in the online mode.

Exceptions:

- Configuring the platform
- Configuring the I/Os
- Importing programs / blocks / data types
- Configuring the DAT_FILE_WRITE block

Another feature:

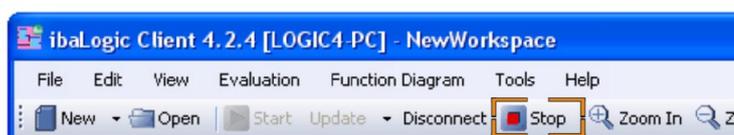
You can end the client in the online mode without stopping the PMAC. When you restart the client, it connects with the PMAC immediately in the online mode.

This is particularly useful when the PMAC is running on another platform (another PC or PADU-S-IT). You can then shut down the ibaLogic computer and, in fact, remove it, while the PMAC continues to run. After rebooting and starting the server and client, the client reconnects automatically with the PMAC running in the online mode.

9.3 Stop the Runtime system

Procedure

1. Click on the <Stop> button in the toolbar.



2. Confirm the "Stop Evaluation..." dialog with "Yes".



Note

This request can be disabled in the ibaLogic options to stop the evaluation in the development and test environment by pressing the <Stop> button or <Shift>+<F5>.

To disable the request, open the options with "Tools"->"Options" and enable the "Program: Stop evaluation" option under [General]->[Messages]->[Confirmations].

Result

The following actions are performed with <Stop>:

- Program execution (PMAC) terminates.
- The background color of the programming field in the client changes to gray.
- The value pads are concealed.
- The program is now in the offline mode.

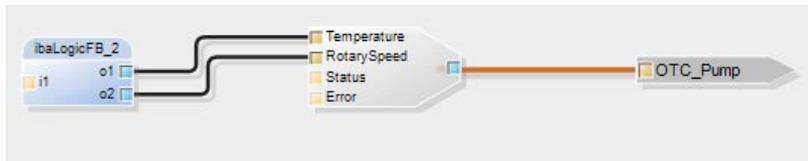


figure 94: Offline mode

9.4 Runtime System – Autostart

9.4.1 Save program on the PMAC

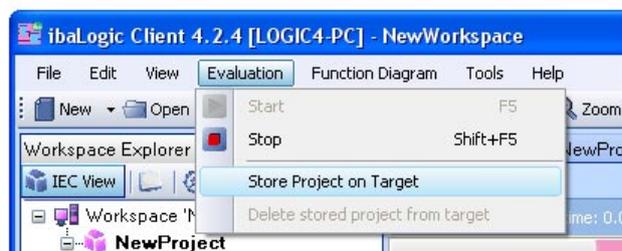
If you wish to start the platform with an executable program, this program must first be saved in the PMAC.

Prerequisite

- ibaLogic is in the online mode.
- Autostart is enabled.

Procedure

- ➔ Select "Evaluation – Store Project on Target" in the main menu.



Result

The project is saved on the target. A file is generated physically. The PMAC will find this file on the platform on startup and execute it.

For more information, please see "Activate Autostart Server, Page 47"

**Note**

Please note that any subsequent program modification must be saved explicitly in the PMAC.

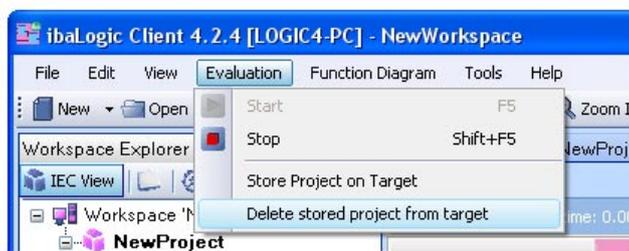
In order to prevent automatic startup of the PMAC, you can first delete the platform memory or modify the autostart options.

9.4.2 Delete Program on the PMAC

This deletes the image file generated earlier physically with the command "Store Project on Target".

Procedure

- Select "Evaluation - Delete stored project from target" in the main menu.

**Result**

The PMAC memory, i. e. the image file created is deleted.

9.5 Connect/disconnect

Disconnect is used when - you have several program modifications to be made one after another - without stopping the program or compiling the steps one by one.

Example

You would like to expand a structure data type that is used many times in the program.

CAUTION

Risk of effective SWITCH/SLIDER in the Disconnect state!

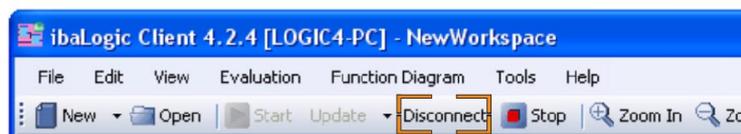
Any SWITCH /SLIDER existing at the time of Disconnect continue to be active on the running PMAC in order to have an influence on the system despite this. This behavior permits operations on the running PMAC despite DISCONNECT.

If any of these SWITCHES or SLIDERS is removed and added again with the same name, this becomes effective again immediately on the running PMAC.

If you delete an FB or MB in the Disconnect state and add it again with the same name, identical inputs and outputs can be visualized again immediately. The values relate to the block running currently in the PMAC and not to the block that has been created afresh. The block that has been created afresh can, for example, have totally different contents. The new layout is accepted only after compiling and loading the program.

Procedure

- ➔ Click on button <Disconnect> in the toolbar.



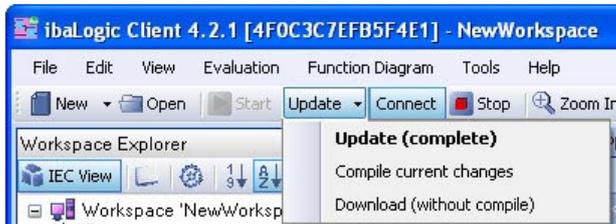
The following actions are performed thereafter:

- Program execution (PMAC) is not terminated, but instead, continues unaffected.
- The background color of the programming window in the client changes to green.



- The value pads continue to be displayed and updated.
- The <Disconnect> button is changed to <Connect>.
- The <Update> button is enabled.
- ➔ You can now replace the old type one by one with the new one in all blocks and OTCs in which this data type is used. The modifications, as you do this, are neither compiled nor loaded in the PMAC.

- After completing your modifications, you can now specifically compile and load them in the PMAC. You can do this either in one shot (complete update) or in separate steps. Select <Update> in the toolbar.



In doing so, you do not quit the Disconnect mode.

- You can exit the Disconnect mode by clicking on <Connect>.

Result

As a result, all changes get compiled and are loaded in the PMAC.

10 Platforms

Before you start configuring the interfaces to the peripherals or to other systems, you have to set up the base hardware on which the ibaLogic runtime system (PMAC) needs to run.

At present, there are two device classes available for the platform:

WinXP

The PMAC runs on a Windows PC on which the other ibaLogic components are running.

For more information, please refer to "Operating and Processing Modes, Page 31".

The link to decentralized peripherals and to other systems is established using PCI cards here.

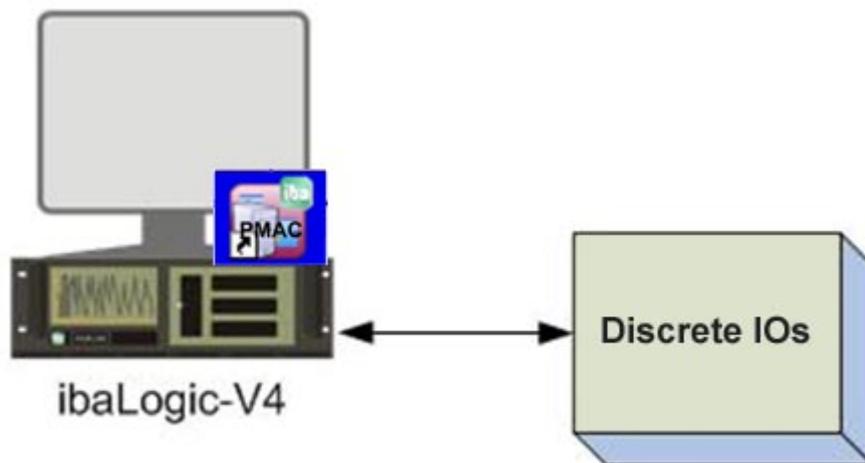


figure 95: Peripheral interface Windows PC

PADU-S-IT

The PMAC runs on an ibaPADU-S-IT station. All other ibaLogic components are always located on one or multiple Windows PCs.

Only the local I/O components (peripheral modules) are available to the PMAC on the ibaPADU-S-IT. There is a bi-directional FO connections and a network connection for TCP/IP coupling available for decentralized peripherals and external systems.

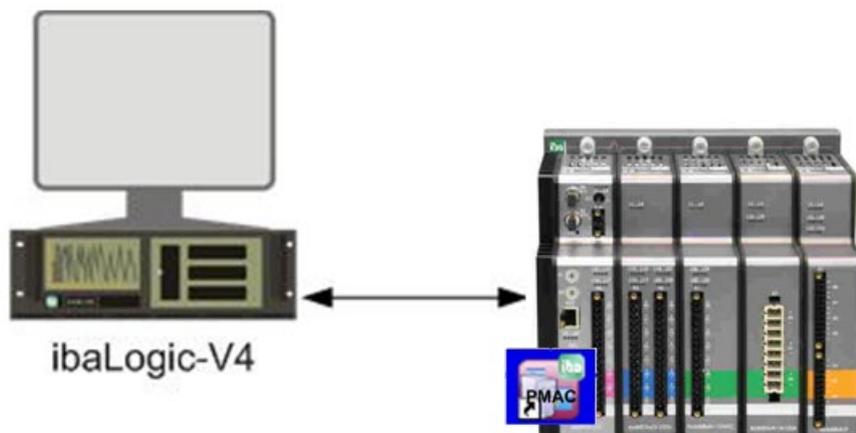


figure 96: Peripheral interface of the PADU-S-IT

After calling the ibaLogic-V4 Client for the first time, the local Windows PC, that is, device class "WinXP" is preset as the platform.

10.1 Configuring the Platform

The dialog box for configuring the platform is available under "Tools" in the menu bar of the Client's ibaLogic.



Note

The platforms are created specifically for each project. All projects of the workspace and the platforms configured in them are available in the "Platform Configuration" dialog box.

Prerequisite

You have opened a project.

Procedure

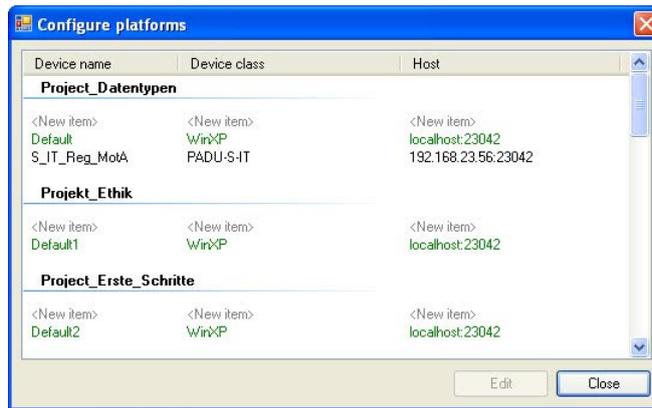
- Select "Tools - Platform Configuration" in the menu.



Alternatively:

open the list next to "Current Platform" in the toolbar and select <Add Platform> or <Edit Platform>.

- Click in the dialog box under the project name on <Add Platform> or on the appropriate platform in order to create a platform or to configure it.



Color	Explanation
Green	Is the active system.
Light gray	Creation of a new platform.
Black	Other platforms available.

- Click on the <Edit> button. The "Edit Platform Configuration" dialog box is displayed.



- Enter a device name or accept the settings specified. Within the workspace, the name must be unique.
- Select the device class WinXP or PADU-S-IT.
- Enter the host IP.
 - If you have selected the WinXP device class, enter "localhost" or "127.0.0.1" when the PMAC runs on the computer or on the server.
 - If the PMAC is located on another computer within the network, enter the host name or the IP address of this computer.
 - If you have set PADU-S-IT as the device class, enter the host name or the IP address of the ibaPADU-S-IT device on which the PMAC should run.
- Confirm your inputs by clicking on the <OK> button.
- Exit the dialog box with <Close>.

10.2 Selecting the Platform

The platform set currently is displayed in the toolbar of the ibaLogic-V4 Client. You can use the selection box to switch to another platform.



Important Note

Please note that the platform is always set for the active project and not for the project being edited.

Procedure

1. Click on the selection box to switch the platform.
2. Select one of the platforms already configured.



Important Note

Please note that the I/O configuration depends on the platform.

After setting up the platform, you have to update the I/O configuration.

Select the "Tools – I/O Configurator" button <Update Hardware >.

11 IO Configuration

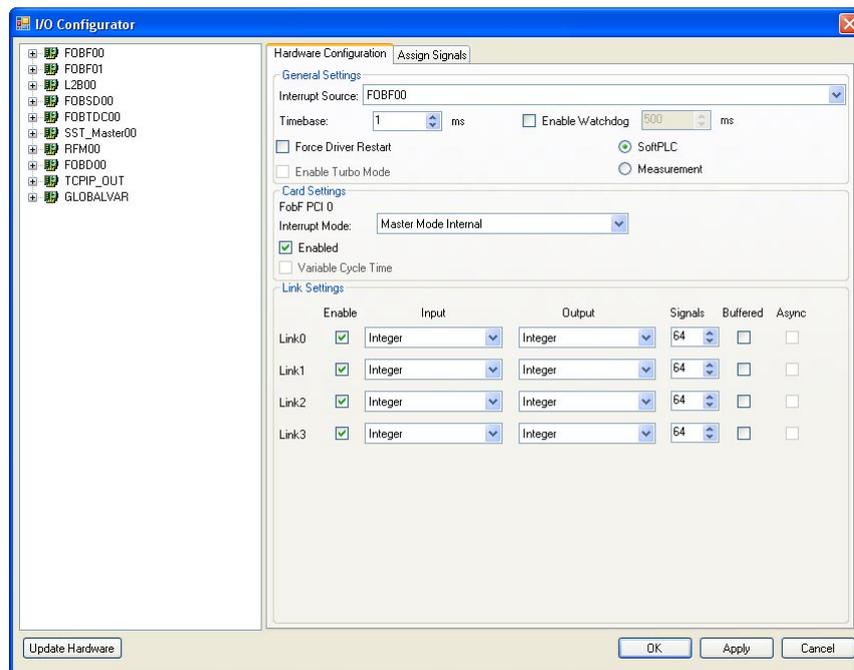
The IO Configurator is the centralized dialog in which you can make all configuration settings relating to the input and output signals as well as certain interfaces.



Note

Exceptions are all those interfaces that are available as function blocks, e.g. TCPIP_SENDRECV block or the in-built OPC interface.

- ➔ Open the IO Configurator using the "Tools - IO Configurator".



The IO Configurator dialog screen is divided into 3 sections:

- Input / output resources available
- Hardware-Configuration
- Signal assignment

Input / output resources available

All hardware and software interfaces detected and supported by the system are displayed in a tree structure on the left side of the dialog screen.

Hardware-Configuration

This is where you can configure special settings for the entire system and for individual cards.

Assign signals

You can work with self-defined and meaningful input and output names in ibaLogic (virtual inputs and outputs). These virtual signals are assigned with the help of signal assignment to the physical inputs and outputs. The virtual signals are divided in groups.

11.1 Resources

The I/O configurator accepts the I/O configuration associated with the active project when it is opened.



Note

The I/O configuration is not saved in the database, but instead, in two XML files in the path, "...\ibaLogic v4\Server\HwMappings". Hence, it is possible to edit the project regardless of the hardware available.

The I/O configuration files are saved during database backup in ZIP files and are reloaded when the database is restored so that after transporting the projects, even the original I/O configuration is available.

When saving the database backup in BAK files, the I/O configuration is not saved.

<Update Hardware> button:

By clicking on the <Update Hardware> button, the hardware that is available to the computer on which the PMAC is running is accepted. This includes the PCI cards supported on the Windows PC platform (see "Hardware Resources, Page 177"). For the ibaPADU-S-IT platform, these are the peripheral modules available there.



Important Note

The platform must be configured prior to editing the I/O configuration, since the settings of the I/O configurator are lost when you switch the platform!



Note

The number of I/O signals permissible depends on the license purchased (Dongle).

Tree structure

Prerequisite

- ❑ You have activated the corresponding link in the hardware configuration and accepted the configuration using the <Accept> button at the lower border of the dialog box.

Procedure

- ➡ You can open the tree structure right up to each individual signal by clicking on the +/- character in front of the names.

The following hierarchy levels are displayed:

Interface → module → inputs/outputs → signals

Interface: Name and index for the card types

Modules: Group according to the physical division, depending on the type of card.

Inputs /
Outputs: Module may have only inputs, only outputs or both

Signals: The names of the hardware signals are formed from the module name, direction, data type and serial number.

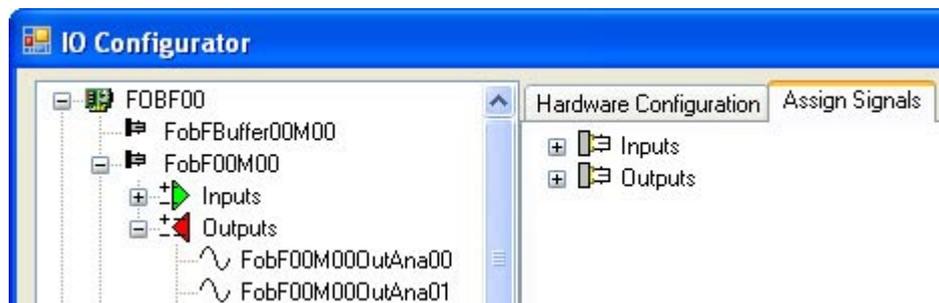


figure 97: Assign signals

11.1.1 Hardware Resources

ibaLogic supports the following interfaces:

WinXP platform and PCI cards

Interface	Cards	Links (Connections)	Protocol
FOBF nn ⁴	ibaFOB-io-S ibaFOB-4i-S ibaFOB-4o-S	1 FOC link 4 FOC links (simplex/duplex)	ibaNet with 2 and 3 Mbit
	ibaFOB-2io-X ibaFOB-4i-X ibaFOB-4o-X	2 FOC links 4 FOC links (simplex/duplex)	ibaNet with 3.3 and 32 Mbit (32 Mbit only for receive)
FOBD ii	ibaFOB-2io-D ibaFOB-4io-D ibaFOB-4o-D	2 FOC links 4 FOC links (simplex/duplex)	ibaNet with 2, 3.3 and 32 Mbit and DMA
FOBSD nn	ibaFOB-SD	1 FOC link duplex (ST)	SIMADYN D
FOBTDC nn	ibaFOB-TDC	1 FOC link duplex (SC)	SIMATIC TDC
L2B nn	ibaCOM-L2B 4/8 ibaCOM-L2B 8/8	4 slaves 8 slaves	Profibus DP Slaves
SST_Master nn	SST-PCB3	max. 125 slaves	Profibus DP Master
RFM nn	VMIC-5565 VMIC-5575	1 duplex FOC 1 coaxial	Reflective Memory

figure 98: WinXP platform

nn = Card numbering 00 to 03 (max. 4 cards of one type are allowed)

ii = Card numbering 00 to 07 (max. 8 cards of type FOB-D are allowed).

PADU-S-IT platform

Interface	Peripherals
PADU-S-IT	Local peripherals, i. e. the interface modules on the PADU-S module rack or frame. Please refer to the PADU-S modules for this purpose.



Other Documentation - PADU-S-IT platform

Please look up the ibaPADU-S-IT manual for information on this.

⁴ Cards for this interface are available only for old systems.

11.1.2 Software Resources

ibaLogic supports the following protocols that are based on Ethernet:

Display	Protocol
TCPIP_OUT	TCP/IP protocol ibaLogic to PDA: WinXP class: max. 16 telegrams for every 32 real values and 32 binary values PADU-S-IT class: max. 16 telegrams for every 32 real values and 32 binary values



Note

Only outputs are available.

11.1.3 Global System Variables

The following global system variables are provided:

Interface	Variables
GLOBALVAR	Global input variables: LAST_DRIVER_ERROR Last error occurred on the driver displayed as hex code. WATCHDOG_BITE FALSE by default. If this value becomes TRUE, the configured watchdog of ibaLogic has responded and switched off the outputs of the cards. For more information on "Watchdog", please refer to "General Settings, Page 180". SYSTEM_UTC_TIME Current system time in UTC format (UniversalTimeCoordinated). DONGLE_NUMBER Dongle number of the iba dongle plugged in on the PC or the serial number of the PADU-S-IT. ACQ_RESTART_COUNT Counter for internal driver restart. This is used to detect system overloads in the "Buffered Mode"

11.2 Hardware Configuration

The hardware configuration dialog screen is called via the "Hardware Configuration" tab.

There are 3 sections available in the hardware configuration in which you can configure the following settings.

- General Settings for ibaLogic
- Card Settings
- Link (Connection) Settings



Note

You can make modifications only if the evaluation has not yet been started (Gray background in the design area).

11.2.1 General Settings

The general settings are applicable to the ibaLogic runtime system and to all interface cards.

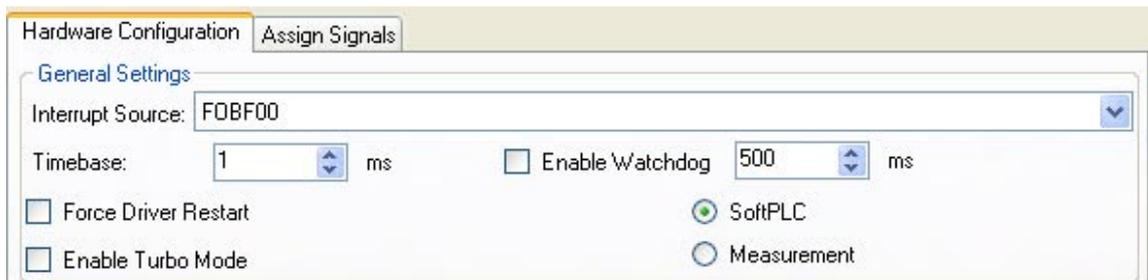


figure 99: General Settings

Interrupt source

The iba modules available are displayed in this list field or selection list. Select any module from this list that should work as the interrupt source with respect to the PCI bus.

If there is no I/O card in the system, ibaLogic clock uses a clock and the field is empty.

Only PADU-S-IT is provided as an option under the PADU-S-IT platform.

Time base

The time base is the smallest cycle time that can be used. Please note that the configurable task intervals cannot be less than this time base.

Activate Turbo Mode

The turbo mode can be activated if a multiprocessor PC is being used. The spare performance capacity of the system can be enhanced considerably with this, since one of the processors is responsible exclusively for the execution of the PMAC and the other one handles the customary Windows management. This should be used particularly for control and regulation functions (Software PLC).

Detailed description see "Time behavior, Page 230".

Soft PLC

This mode is suitable for control and regulation functions. This ensures in ibaLogic that only the latest signal states are processed. In contrast to the measurement mode, it is not decisive if samples are lost. Current data from the latest I/O transfer cycle are used for the evaluations.

Detailed description see "Time behavior, Page 230".

Measurement

This mode ensures that ibaLogic does not lose any input sample. This is also true when individual tasks within ibaLogic need to be superseded. The runtime system of ibaLogic ensures that the data are made available equidistantly in the task interval configured. If tasks get superseded, the system makes up for the cycles.

Detailed description see "Time behavior, Page 230".

Activate Watchdog

When this function is activated, a timer is started up in the cards available, and this timer is triggered by ibaLogic when there is a write operation at the outputs. If there is no write command (= trigger) from ibaLogic within the time configured, the card automatically sets all outputs to 0.

Since ibaLogic writes to the outputs cyclically, any trigger of the watchdog points to an application that has paused or is overloaded (e.g. b programmed endless loop).

Force Driver Restart

This function is particularly important for SST cards and reflective memory cards. These cards are also reset with the driver restart and, with it, the modified external configuration is accepted. Setting this option leads to restart with <OK>. This option is again reset automatically.

11.2.2 Card Settings

Please mark the appropriate interface in the tree structure on the left for the settings. Only those settings are described here that are applicable to (almost) all cards.

Interrupt mode

This field is provided only for iba cards.

The following modes are available for selection:

Master Mode Internal:

Master Mode Internal should be set for only one card. This uses the internal timer to generate a synchronization signal that is distributed via a flat ribbon cable to the other cards.

Master Mode External:

In contrast to Master Mode Internal, the synchronization signal is not generated in the card, but instead, derived from the cycle of the FOC telegram incoming at Link0. This mode is meaningful only if ibaLogic needs to be synchronized with an external cycle, e. g. with variable interrupt time in buffered mode (e. g. flatness measurement or for bus-synchronous measurement with the Simolink monitor. For further information, please refer to "Buffered Mode, Page 192".

- Slave Mode:
This mode must be set for all other cards.



Note

Select this as the interrupt master if there is a card of type FOBSD or FOBTDC in your configuration. Otherwise, select one from the FOB-X or FOB-D type of cards. If neither of these cards is present, you can use the FOB-S or L2B card.

- Enabled

You can use only cards that have been enabled.

A card must be deactivated if an ibaPDA Server is running on the same computer that uses this card. A card cannot be used by ibaLogic and ibaPDA simultaneously.

Other settings are specific to each card and described in section "PCI Interfaces (Windows PC), Page 190".

11.3 Signal assignment

In order to use the physical inputs and outputs, you must assign them to the virtual signals in the program.

There are two methods for assigning signals:

- from the hardware signal to the program signal (seen from the hardware).
- from the program signal to the hardware signal (seen from the program).

It is also possible to use a mixed method between these two basic methods.

11.3.1 Method as seen from the hardware

At the start of programming, you already know the external interfaces, e.g. the assignment of the FOC telegrams or that of the Profibus slaves. From these physical signals you generate virtual signals that you can use subsequently in the program.

You can accept the names generated automatically or assign your own base names. The direction tag, type and index are added automatically. Finally, each signal name can be changed separately.

After acceptance, the virtual signals are visible in the navigation area under "Inputs - Outputs". Here, too, you can modify the signal names provided that they are not yet used in the program, i.e. dragged from the navigation area to the borders of the design area.

11.3.1.1 Example: Assignment of all signals of a module of an ibaFOB-io-S card

The FOB card is listed, among others, on the left side.

The inputs and outputs are not yet assigned in the "Assign Signals" tab.

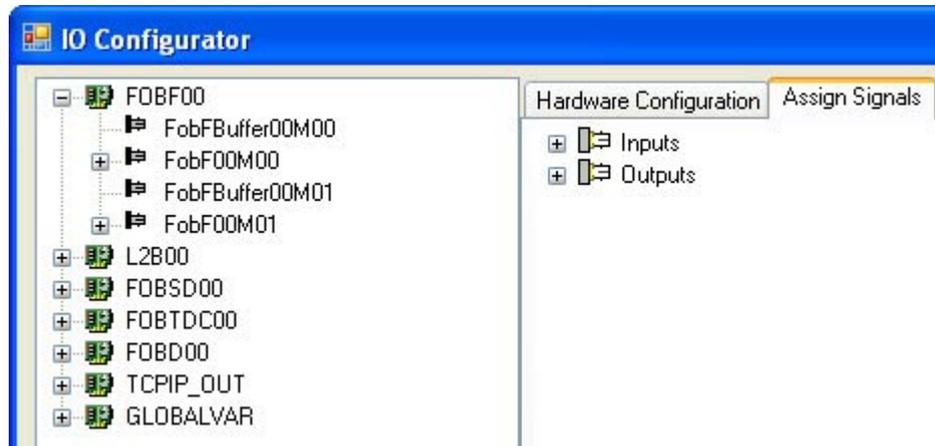


figure 100: Assignment of signals of the ibaFOB-io-S card

Procedure

1. Open the tree of the FOB card on the left side.
All activated links of the card and their designations are displayed.

You can assign the entire FOBF00M00 module (equivalent to link 0 on the FOB card).

2. Drag the FOBF00M00 module and drop it on the right side on the inputs or outputs.
The "Group Properties" dialog box is displayed.



Remark

ibaLogic creates a group with the group name under the inputs and forms a virtual name for each hardware signal. You can either leave the generation of the names entirely to ibaLogic or specify them yourself.

During the generation, the composition of the current signal name is displayed. In the example given above, the name "FOBF00M00InAna01" is composed as follows:

FOBF00M00	Base name (same as the group name)
In	Direction tag
Ana	Type number
01	Serial number

Depending on the direction that you have chosen in the selection box, all signals of this module are created in the group displayed under inputs and / or outputs.

Result

The assignment leads to the following result:

The virtual signal names are displayed to the left from the arrow and the hardware signal names to the right. The group name is identical to the physical module name.

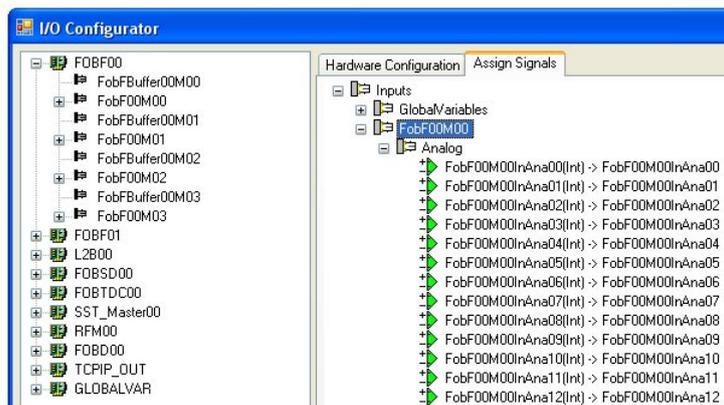


figure 101: Assign signals

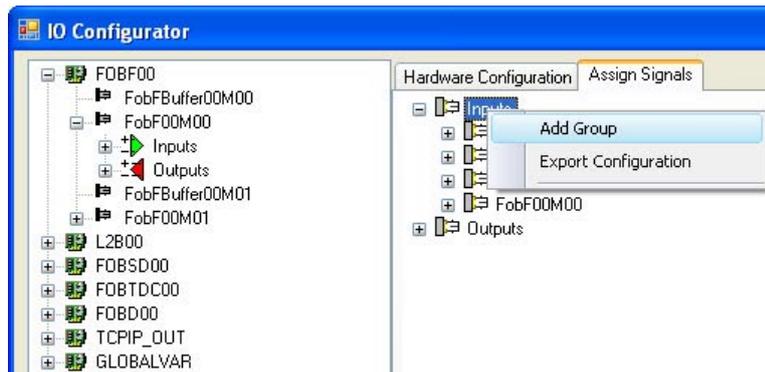
11.3.1.2 Example: Assignment of individual signals of an ibaFOB-4i-S or ibaFOB-4o-S card

If you need only a few signals of a module in the program, you can assign only these signals as required.

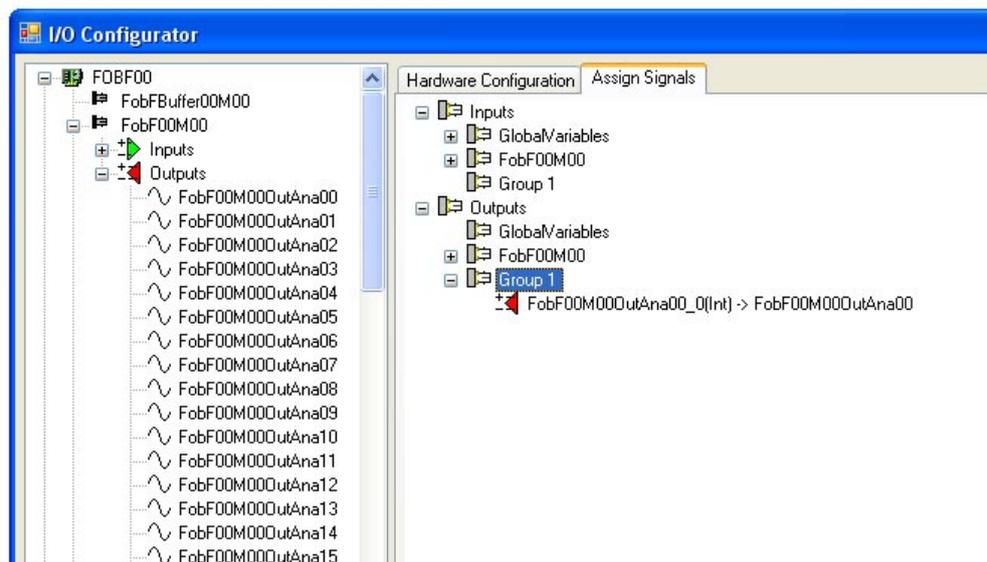
Procedure

Add group

1. Open the context menu by clicking the right mouse button.



2. Select the "Add Group" menu item, assign a group name and leave the "Setting the Group Name" dialog box by clicking OK.
3. Open the signal tree until you can see the individual signals of the module you want to use.
4. Add individual signals to the group defined earlier by drag and drop.



Result

Individual signals are assigned to the group.

11.3.1.3 Change Signal and Group Names

You can change the names of the virtual signals one by one after the assignment.

Procedure

1. You can change the group name by clicking on the right mouse button. Select the "Properties" menu item.
2. You can modify the signal name by double clicking on it. You can also add a description for the signal. The description is displayed in the program as a tooltip.
3. Confirm the settings with <Accept> or with <OK>.

11.3.2 Procedure as seen from the program

At the start of programming the external interface are not yet known, but you would like to commence with programming and find out the inputs and outputs that you need.

1. Define the inputs and outputs, groups and signals in the navigation area. Please see the description in "Configure Inputs and Outputs, Page 80" for this purpose.
2. As soon as you know the physical interfaces at which the IO Signals are available, you can switch to the IO Configurator and assign these signals to the physical interfaces.

11.3.2.1 Example: Signals of an ibaFOB-4io-S card (complete module)

You assign the physical signals of link0 of the FOB card.

Prerequisite

You have defined a group "MotorA" and under this, the input signals "MotorA_N_Ist(Int)", "MotorA-Ta(Int)" and "MotorA_Status(Int)" in the program (in the navigation area of the inputs and outputs).

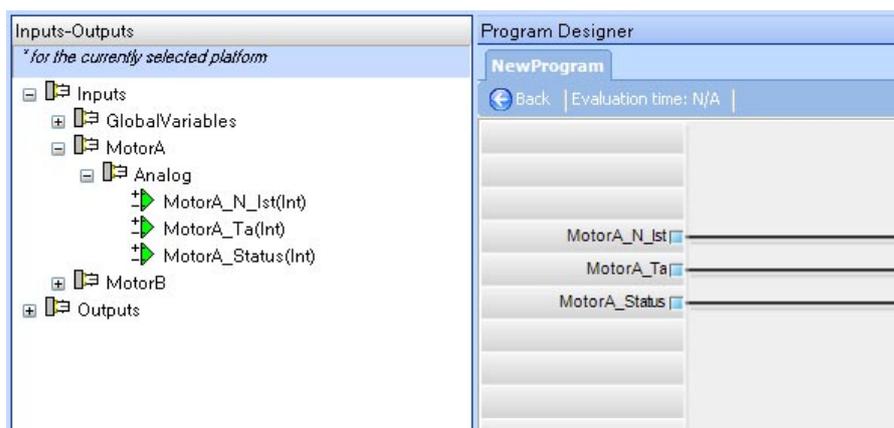
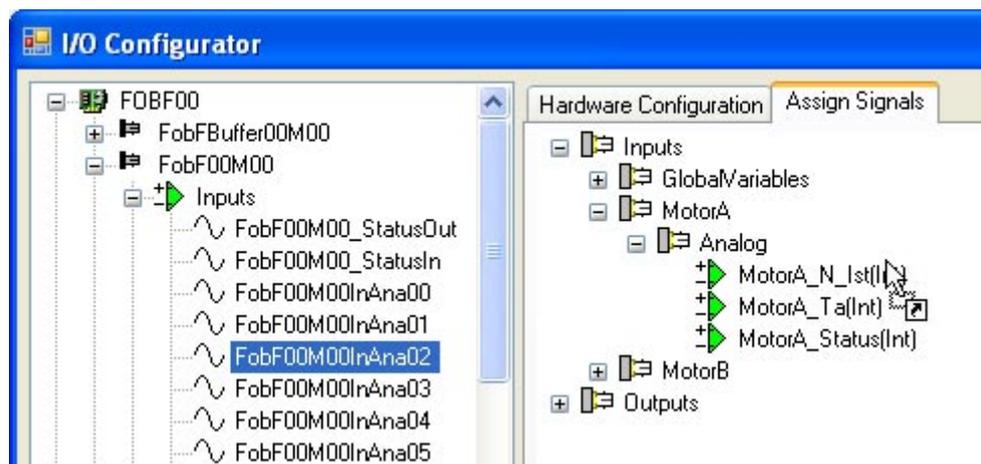


figure 102: "Inputs – Outputs"

Procedure

1. Open the I/O configurator.
2. Update the hardware using the <Update Hardware> button.
3. Activate link0 on the FOB card.
Please make sure that the data type of the link matches that of the signals already defined.
You can see the signals defined in the project in the "Assign Signals" tab.
4. Select a hardware signal from the tree structure on the left.
Drag & drop it on the signal.
You have, thus, assigned the virtual signal to a physical signal.



Result

You can also see the assignment in the design area.

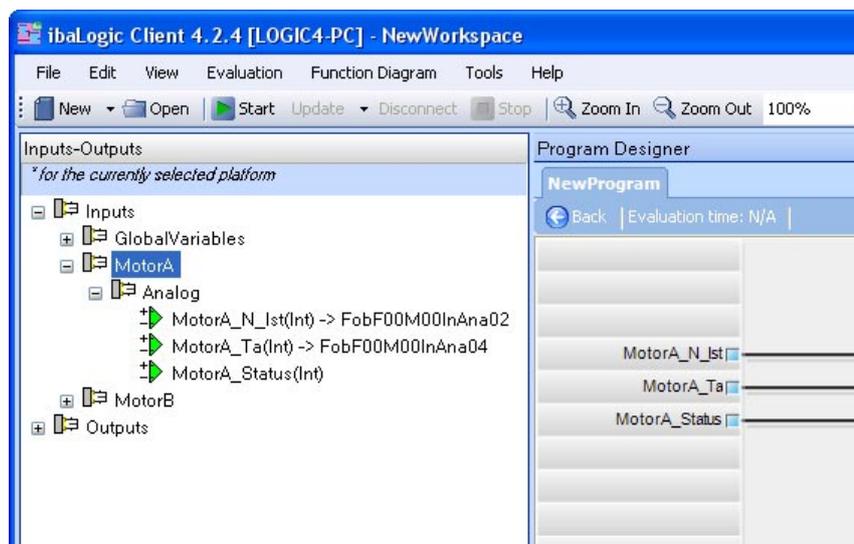


figure 103: Assign signals

11.3.3 Modify Signal Assignment

You can modify existing assignments. Multiple methods are possible.

Method 1

- Simply drag another hardware signal and drop it on a virtual signal that has already been assigned.

The assignment gets modified by doing this.

Method 2

- Drag a hardware signal that is already in use and drop it on a virtual signal. An information message pops up.
- Acknowledge the information message, after which the assignment gets modified and the link to the old signal is deleted.



Note

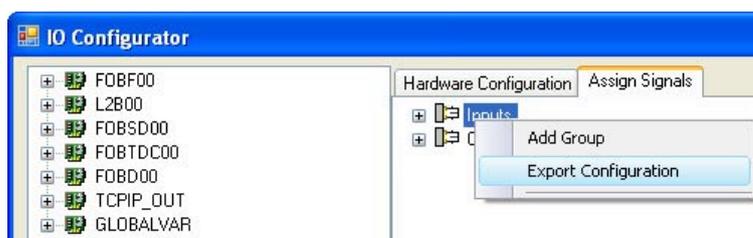
You can assign only one virtual signal to a hardware signal.

11.3.4 Using externally defined signal names

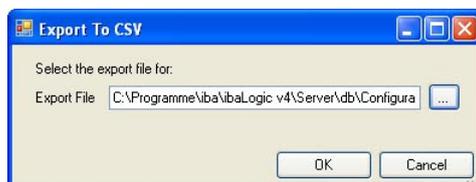
Using the export / import function, you can also use signal names that are available in external documents, e. g. in a spreadsheet program.

Procedure

- Specify the hardware signals.
- Export the I/O configuration using the context menu of the inputs and outputs.



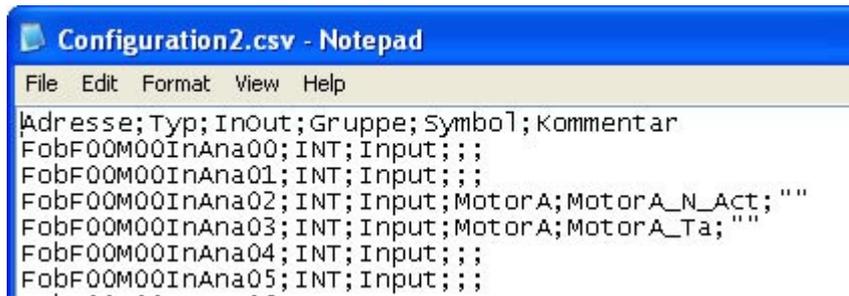
- Select "Export Configuration" in the menu. A dialog box is displayed.



- Specify the target path and file name.

Result

You get a CSV file that you can open using an ASCII editor or a spreadsheet program.



```
Configuration2.csv - Notepad
File Edit Format View Help
Adresse;Typ;InOut;Gruppe;Symbol;Kommentar
FobF00M00InAna00;INT;Input;;;
FobF00M00InAna01;INT;Input;;;
FobF00M00InAna02;INT;Input;MotorA;MotorA_N_Act;""
FobF00M00InAna03;INT;Input;MotorA;MotorA_Ta;""
FobF00M00InAna04;INT;Input;;;
FobF00M00InAna05;INT;Input;;;
```

figure 104: CSV file in ASCII editor (Notepad)



Important Note

You can see the hardware signals defined in the columns, Address, Type and InOut.

Do not modify these.

You can edit the virtual signals under the columns, Group, Symbol and Comments, or use copy & paste to accept them from the external document.

Please note that the names in the "Symbol" column conform to the IEC standard.

- Exit the IO Configurator to import the data.
- Select "File - Import - Signal mapping" from the main menu.

11.4 PCI Interfaces (Windows PC)

This section describes the special settings of the interface cards.

For more information please refer to "Hardware Resources, Page 177".

11.4.1 Connection to the "iba World"

The following PCI cards are available for FOC connection to decentralized iba peripherals (PADUs) and to the iba system interface cards. You will find them in the resource tree under the **FOBFnn** or **FOBDnn** interface type.

- ibaFOB-S⁵ (FOC link, 3 Mbit Protocol)
- ibaFOB-X⁶ (FOC link, 32 Mbit Protocol)
- ibaFOB-D (FOC link, 2 Mbit / 3 Mbit / 5 Mbit / 32 Mbit Protocol)

For each type of card, there are variants for the number of links being 1, 2 or 4 for the input and output.

11.4.1.1 Card Settings

When you mark the **FOBFnn** or **FOBDnn** interface in the tree on the left, the associated card settings are displayed on the right.

- Interrupt mode, see "Hardware Resources, Page 177"
- Enabled, see "Hardware Resources, Page 177"
- Variable cycle time



figure 105: Card settings

Check box	Explanation
Enabled	You can use only cards that have been enabled.
Variable cycle time	This mode has not been realized.

⁵ Cards for this interface are available only for old systems.

⁶ Cards for this interface are available only for old systems.

11.4.1.2 Link Settings

The channel configurations are displayed in the section on "Link settings". The number of channels varies depending on the variant of the ibaFOB-io card used.

Settings	Explanation
Enable	You can enable or disable individual links here.
Input format	Set the data format for the incoming optical fiber cable telegrams here. Select INTEGER, REAL or S5REAL depending on the devices connected. If you are using the FOB-X or the FOB-D card, other data formats for the 32 Mbit FOC protocol are provided as options.

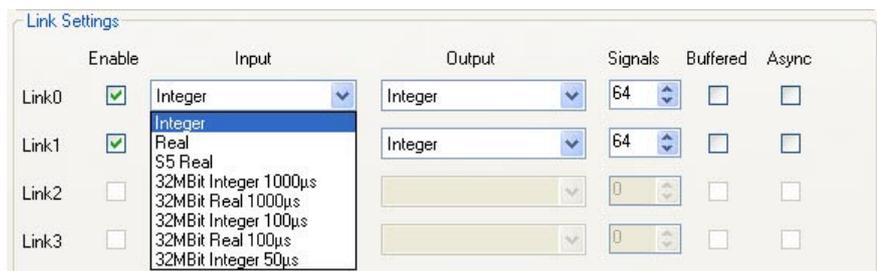


figure 106: Link settings

The data type must match the type or the setting configured on the device connected.

Device	Protocol	Data type/telegram type
ibaPADU-8 ibaNet750	3 Mbit	Integer
SIMATIC TDC/LO6	32 Mbit	32 Mbit Real 100 µs
ibaLink-SM-64-i-o	3 Mbit	Integer, Real or S5Real, depending on the switch setting on the module.
ibaLink-SM-64-SD16 ibaLink-SM-128V-i-2 ibaBM-DPM-S-64	3 Mbit	Integer or Real, depending on the setting on the module
iba-PADU-S-IT	32 Mbit	depending on the setting in ibaPADU-S-IT
ibaBM-DPM-S (Profibus) ABB AC800PEC	32 Mbit	32 Mbit Real 1000 µs
ibaLink-VME	32 Mbit	depending on the card setting



Note

Applicable to FOB-X cards: If a 32 Mbit protocol is used on the input side, the associated output link cannot be used.

Settings	Explanation
Output Format	Data type of the FOC telegram in the output direction. The setting depends on the device connected (see input format).
Signals	<p>Here, ibaLogic enters the maximum number of signals possible depending on the data format selected. You can reduce the number of signals for your requirement.</p> <p>The number of signals entered applies to analog values and digital values in the input and output direction.</p> <p>An "FobFnnMxx" or "FobDnnMxx" module having "n" analog signals of the type configured and "n" binary signals (nn = card index, xx = link number) is created for each link enabled after you press <Accept> in the resources tree under the appropriate interface:</p>
Buffered mode	With this, you can enable a mode for this link in which the data received is also provided as arrays. See the following description for this purpose.

11.4.2 Buffered Mode

11.4.2.1 Applications

The buffered mode is necessary for the following applications:

- The least possible interval time of ibaLogic-V4 is 1 ms. If signal values need to be sampled with a cycle time of less than 1 ms, the signal values must be recorded in buffered mode. Above all, this must be used in the context of the following modules:
 - FOB-D and FOB-X in 32 Mbit mode
 - Connection to ibaPADU-S-IT (in I/O mode)
 - Connection of the local peripherals for the PADU-S-IT platform
- Even for peripheral signals that arrive with a cycle time of 1 ms, but which cannot be evaluated in the 1 ms interval on account of the large number of signals and the computing power required.

The values read in are buffered and provided to the program as array resources. The purpose of this mode is to relieve the program from the evaluation of the individual signals if this is not necessary.

Example:

Signals are required merely for FFT analysis. It is then not possible (with a sampling time < 1 ms) and also not meaningful to sample individual signals and to collect them in arrays in order to perform an FFT analysis on them. The arrays are generated by the buffered mode with the proper size so that no computing time is required by the program for handling individual values.

The buffered mode has the following features:

- ❑ Data from the FOB link is sampled by the driver at a rate determined by the time base configured.
- ❑ The data sampled is collected in a cyclic buffer for each signal. The size of the cyclic buffer and the sampling rate for filling it up are configured by the program in the output resources (see below).
- ❑ Each signal is made available to the program in the form of an array having a length of 256. The program runs in a slower task interval and reads the complete arrays.
- ❑ The program, thus, cannot process individual samples, but instead, only arrays of samples, e. g. for evaluating an FFT analysis or for archiving.
- ❑ Example: Despite a sampling rate of 1 ms, and a task interval of 50 ms, you can still perform an FFT analysis with 128 samples.
- ❑ In parallel with the buffered data, individual signals can, for example, also be generated for other programs.



Note

Buffered mode is possible only for the "Measurement" sampling mode.

11.4.2.2 Input Resources

An FobFBuffer nn M xx or FobDBuffer nn M xx mode is created (nn = card index, xx = link number) for each link enabled in buffered mode after pressing <Accept> in the resource tree under the appropriate interface.

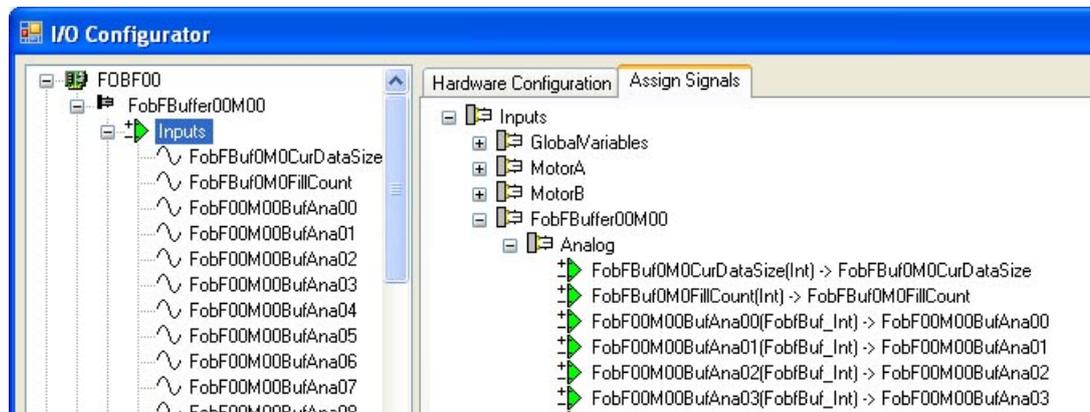


figure 107: Assign input signals

Input signals	Explanation
CurDataSize	Feedback of the buffer size configured in the output signal.
FillCount	Counter that is incremented when the input array has been filled up to the length "Datasize".
BufAnaii	Array of type integer or real having a length of 256.
BufDig00	Array of type Boolean having a length of 256.
Only for "variable cycle time"	
CurCycleTime	Feedback of the interrupt cycle time configured in the output signal.

11.4.2.3 Output Resources

The resources are provided in the module type FobFBuffer nn M xx or FobDBuffer nn M xx (nn = card index, xx = link index).

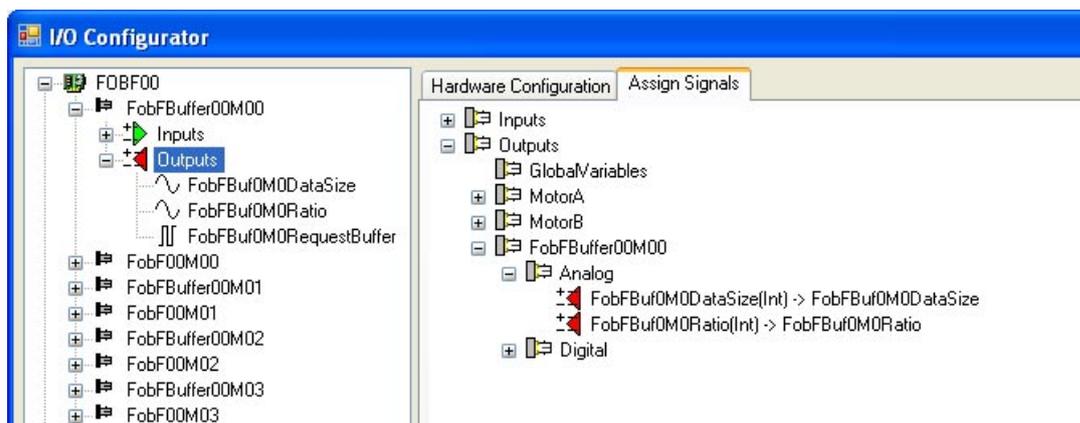


figure 108: Assign output resources

Output signals	Explanation
Data size	Number of signals to be measured that are entered by the driver in the buffer until the counter is incremented ("Filling level"). Values up to 256 are permissible.
Ratio	Integral multiple of the time base, with which the buffers are filled up. For example, ratio=2 means that only every 2nd value of signal sampled is entered in the buffer.
RequestBuffer	Controlling the sampling. The buffers are filled up only if the output is "TRUE"
Only for "variable cycle time"	
CycleTime	Variable interrupt time. Specification in microseconds. Permissible values: 1000 ... 10000.
SetCycleTime	Signal for accepting the cycle time (rising edge).

11.4.3 ibaLogic as Profibus Slave

The Profibus is organized strictly according to the master-slave principle. In accordance with the DP-V0 standard, communication takes place only between the master and slaves, whereby the link is set up and monitored by the master. ibaLogic can be both a Profibus master and a Profibus slave.

For example, for a link to a SIMATIC S7 system that is the master, ibaLogic must work as a slave. The Profibus slave card, ibaCOM-L2B is used for this purpose.

The following interface cards are arranged under the **L2Bnn** type of interface:

- ibaCOM-L2B 4/8
(Profibus DP slave card with a jack, 4 slaves)
- ibaCOM-L2B 8/8
(Profibus DP slave card with two jacks, 8 slaves)

11.4.3.1 Card Settings

When you mark the L2Bnn interface in the tree on the left, the associated card setting are displayed on the right.

Settings Processor 0		
Enable	SlaveNo.	Mode
<input checked="" type="checkbox"/>	10	Integer I/O
<input checked="" type="checkbox"/>	11	Integer I/O
<input checked="" type="checkbox"/>	12	Integer I/O
<input checked="" type="checkbox"/>	13	Integer I/O

Settings Processor 1		
Enable	SlaveNo.	Mode
<input checked="" type="checkbox"/>	14	Integer I/O
<input checked="" type="checkbox"/>	15	Integer I/O
<input checked="" type="checkbox"/>	16	Integer I/O
<input checked="" type="checkbox"/>	17	Integer I/O

figure 109: Card Settings for ibaCom-L2B

- Interrupt Mode, see "Hardware Resources, Page 177"
- Enabled, see "Hardware Resources, Page 177"

11.4.3.2 Settings for bus interface 0/1

The L2B card has one or 2 Profibus DP interfaces. You can define 4 slaves for each interface.

Settings	Explanation
Enable	You can enable or disable individual slaves here.
Slave no.	Assign a separate station number to each slave that is enabled.
Mode	Select a telegram format for each enabled slave that matches with the Profibus master configuration. For this purpose, iba provides a number of GSD files that correspond to the telegram formats available here for selection:

Telegram format	iba GSD file	Contents	Data direction
Integer_In	iba_0F01	32 integer and 32 binary signals	Master → ibaLogic
Real_In	iba_0F02	32 real and 32 binary signals	Master → ibaLogic
S7Real_In	iba_0F04	28 real and 32 binary signals	Master → ibaLogic
Integer_inOut	iba_0F08	32 integer and 32 binary signals	Master ↔ ibaLogic
Real_InOut	iba_0F09	32 real and 32 binary signals	Master ↔ ibaLogic
S7Real_InOut	iba_0F0B	28 real and 32 binary signals	Master ↔ ibaLogic

A module "L2BnnMyySxx" is created for each slave enabled after pressing <Accept> in the resource tree under the appropriate interface, and this module has the analog signals of the type configured and 32 binary signals (*nn* = card index 00-03, *yy* = interface number 00-01), *xx* = slave number 00-03).

11.4.4 ibaLogic as Profibus Master

If you would like to address, for example, an ET2000 station from ibaLogic, it must work as the master. A Profibus master card must be installed in the ibaLogic PC for this purpose.

The following interface card is plugged in under the **SST_Masternn** type of interface:

- SST-PB3-PCU (one channel, PCI)
- SST-PB3-PCU-2 (two channels, PCI)
- SST-PB3-PCIE-1 (one channel, PCI Express)
- SST-PB3-PCIE-2 (two channels, PCI Express)



Note

Using the SST card requires a license.

11.4.4.1 Brief Description

Since this card is the product of another manufacturer, its configuration and customization varies from the scheme for the iba cards.

In general, you must generate a configuration for the Profibus that contains the parameters required for all the stations connected to the Profibus and for the communication. The program for creating the configuration (SST Profibus Console) generates a binary parameter file (.bss) as the result. This must be loaded by the ibaLogic in the I/O configurator and transferred to the SST card.

11.4.4.2 Card Settings

When you mark the SST_Master nn interface in the tree on the left, the associated Card Settings are displayed on the right.

The screenshot shows a 'Card Settings' window for a device 'PFB3-PCI-0000'. The 'Enabled' checkbox is checked. Under 'Configuration Channel 0', the 'Enabled' checkbox is checked, and the 'File' field contains 'C:\Program Files\iba\SST configs\Config08.bss'. There are four radio button options: 'Swap Off' (selected), 'Swap Words', 'Swap per Datatype', and 'Swap DWords'. Under 'Configuration Channel 1', the 'Enabled' checkbox is unchecked, and the 'File' field is empty. There are also four radio button options for Channel 1, all of which are unselected.

figure 110: Card Settings

"Enabled"

You can use this option to enable or disable the card.

11.4.4.3 Configuration

"File"

This is where you specify the file generated using the CONSOLE program mentioned above. You can click on to its right to open a browser with which you can select the file in the folder structure.

"Swap modes"

Depending on the device connected, you may have to swap, i. e. interchange the high and low parts of the data type so that the data can be read in and processed in ibaLogic according to the IEC standard.

Explanation of the swap methods (each alphabet means one byte, blanks have been added only for the sake of clarity):

Swap method	Output	Becomes
per Datatype	"ABCD EF G H IJKL"	"DCBA FE G H LKJI"
Words	"ABCD EF G H IJKL"	"BADC FE H G JILK"
Dwords	"ABCD EF G H IJKL"	"DCBA HG F E LKJI"

11.4.4.4 Peculiarities with signal assignment

If you have enabled the configuration settings file of the card with <Accept>, a module having the name of the SST card "PFB3-PCI-000x" is created in the resource tree under SST_Master. The following signals for each possible Profibus slave are visible under this:

Direction	Signal	Meaning
Input	SST nn InStatus yyy	Status and diagnostics information for telegrams received from the slave yyy . Data type: "SSTSTATUSSTRUCT"
Output	SST nn OutStatus yyy	Status and diagnostics information for telegrams transmitted to the slave yyy . Data type: "SSTSTATUSSTRUCT"
Input	SST nn StructIn yyy	Data received from the slave yyy Data type "SST_Struct"
Output	SST nn StructOut yyy	Data transmitted to the slave yyy Data type "SST_Struct"

nn = card index 00-03, yyy = Profibus slave number from 002-127

Now, you can generate the virtual signals for the complete module (see section "Signal assignment, Page 182") or only for the status, input and output signals individually for the Profibus slaves that you have configured and customized.

**Note**

The signals generated here have the structure data types.

The status structure generated internally, "SSTSTATUSSTRUCT" consists of a number (ErrorCode) and a text string (ErrorString).

The "SST_Struct" data type is a placeholder for the structures of the user data telegrams that you have to define. This structure must match the Profibus telegram to be transmitted or received exactly, as you have defined in the Profibus configuration (Profibus Console) for each station. You can refer to the manual of the L2B cards for the structure of the Profibus telegrams (L2B).

For more information, please refer to "Data types, Page 278".

**Important Note**

An example of the connection of the Profibus master card is documented and included in the CD supplied.

11.4.5 SIMADYN D / SIMATIC TDC Connection

iba has developed two PCI cards for connection to these systems and these cards differ from one another merely in the FOC interface technology and protocols.

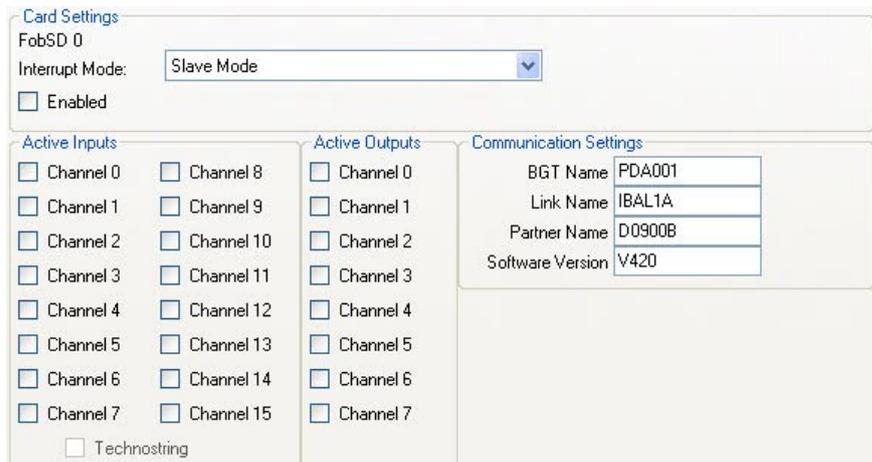
- ❑ The ibaFOB-SD card is plugged in at the **FOBSD** type of interface. This enables connection to the world of SIMADYN D via the rack coupling modules CS12, CS13 and CS14 as well as to the SIMATIC TDC rack using the CP53M0 communication module.
- ❑ The ibaFOB-TDC card is plugged in at the **FOBTDC** type of interface. This enables connection to the world of SIMATIC TDC via the GDM (Global Data Memory, CP52IO interface module).

**Note**

The settings for the FOBSD and FOBTDC modules are identical. Hence, the settings are explained here using the ibaFOB-SD card as an example.

11.4.5.1 Card settings

When you mark the FOBSD interface in the tree on the left, the associated Card Settings are displayed on the right.



Card Settings

FobSD 0

Interrupt Mode:

Enabled

Active Inputs

Channel 0 Channel 8 Channel 0

Channel 1 Channel 9 Channel 1

Channel 2 Channel 10 Channel 2

Channel 3 Channel 11 Channel 3

Channel 4 Channel 12 Channel 4

Channel 5 Channel 13 Channel 5

Channel 6 Channel 14 Channel 6

Channel 7 Channel 15 Channel 7

Technostrating

Active Outputs

Channel 0

Channel 1

Channel 2

Channel 3

Channel 4

Channel 5

Channel 6

Channel 7

Communication Settings

BGT Name

Link Name

Partner Name

Software Version

figure 111: Card Settings for ibaFOB-SD/TDC

- Interrupt Mode, see "Hardware Resources, Page 177"
- Enabled, see "Hardware Resources, Page 177"

11.4.5.2 Link settings

Active Inputs

Select one or more inputs here specifically from a total of 16 channels.

Please note that there must be a transmit telegram present in the SIMADYN D or SIMATIC TDC for each input channel that you select.

One transmit telegram contains exactly 32 real values (Data type NF for SIMADYN D) and 32 binary values (1 DWORD or V4 value).

The following parameters must be configured on the transmitter module:

- AT (Channel name):
- MOD (Channel mode):
- LEN (Telegram length):

Active Outputs

Here, you can specifically select one or more outputs from a total of 8 channels.

Please note that a receive telegram must be present in the SIMADYN D or SIMATIC TDC for each output channel that you select.

A receive telegram contains exactly 32 real values (Data type NF for SIMADYN D) and 32 binary values (1 DWORD or V4 value).

The following parameters must be configured on the receiver module:

- AR (Channel name): PDAMxDAT (x = 0 ... 7 for channel 0 to 7)
- MOD (Channel mode): R (for Refresh)
- LEN (Telegram length): 132 (only for the CRV_P receiver module)

An "FOBSD nn CH xx " (nn = card index, xx = channel index) module containing a total of 32 real and 32 binary signals is created for each channel enabled after pressing <Accept> in the resource tree under the appropriate interface.

Technostring

The "Technostring" function has not yet been released.

11.4.5.3 Communication Settings

- BGT Name:
The PC must be defined to the SD/TDC environment using a name consisting of six characters, e. g. "IBA001".
- Link Name:
The ibaFOB-SD uses this name to register with the communication partner SIMADYN D or SIMATIC TDC. This name must be unique within the CS14 or CP53M0 communication island, i. e. no other Siemens or iba modules should register with the same name. The default name is "IBAL1A".
- Partner Name:
Enter the name configured in the SIMADYN D or SIMATIC TDC for the communication partner module here.
For SIMADYN D, it is the name of the CS14 module, e. g. "D0500B", and for SIMATIC TDC (GDM), it is the name of the GDM module, usually D01_P1.
- Software Version:
Enter the software version of the SIMADYN D or SIMATIC TDC software here: e. g. for STRUC V420", and for CFC "V610".



Note

No communication takes place if these settings are incorrect.

11.4.6 Reflective Memory

You can access VME bus-based third-party systems (e.g.: GE FANUC, Converteam HPCi) using reflective memory cards.

The following interface cards are plugged in under the RFM type of interface:

- VMIC PCI-5565PIORC:
(Reflective memory card, 64 or 128 MByte)
- VMIC PCI-5588 among others:
(Reflective memory cards of older design)

11.4.6.1 Brief Description

Since this card is the product of another manufacturer, its configuration and customization varies from the scheme of the iba cards.

As the memory of an RFM card does not have any homogeneous data range, but contains ranges with different data types, it cannot be used the same parameters as that for iba-FOB cards.

You have to define the position and structure of the data used in an external parameter file. This is loaded in the ibaLogic I/O configurator and signals in the desired data types are generated from there.

11.4.6.2 Card Settings

When you mark the RFM interface in the tree on the left, the associated Card Settings are displayed on the right.

figure 112: Reflective memory cards

- Enabled

You can use this option to enable or disable the card.

11.4.6.3 Configuration

- Swap mode

The swap mode is only available for the older RFM cards which directly support this in the hardware.

Newer cards (PCI-5565 or PCIE-5565) no longer support this.



Note

The methods defined here are different from those for the SST card. The designations here are taken over from the VMIC so that they match with the RFM cards connected to them.

Explanation of the swap methods (each alphabet means one byte, blanks have been added only for the sake of clarity):

Swap mode	Explanation
Not	"ABCD EF G H IJKL"
Bytes	"BADC FE H G JILK"
Words	"CDAB GH E F KLIJ"
Words and bytes	"DCBA HG F E LKJI"
based on data type	"DCBA FE G H LKJ"

11.4.6.4 File

Parameter file that contains the description of the signals.

Procedure

1. Generate a template for this data using the <Generate Template> button.
2. Next, open this file using an editor.
3. Enter a line in the following format for each signal:
"Signal name, data type, memory address, bit number, direction, comments"

Format	Explanation
Signal name	Must conform to the IEC standard and be unique, i.e. also input and output signal names have to be different.
Data type	Elementary data type (see "Standard data types, Page 278") BOOL BYTE, WORD, DWORD SINT, USINT, INT, UINT, DINT, UDINT REAL, LREAL
Memory address	Offset within the RFM memory in decimal or hexadecimal. You can switch from one to the other using a line with "#hexval" or "#intval" at the beginning.
Bit number	Relevant only for BOOLEAN data type, otherwise 0
Direction	"INPUT" or "OUTPUT"
Comments	Any text

Example

```
#HexVal
TestSignal1,REAL,0x1000,0,INPUT, test signal input
TestSignal2,REAL,0x2000,0,OUTPUT, test signal output
#IntVal
TestBit_0,BOOL,2048,0,OUTPUT, Bit 0
TestBit_1,BOOL,2048,1,OUTPUT, Bit 1
TestBit_2,BOOL,2048,2,OUTPUT, Bit 2
```

Result

A template file is generated.

11.4.6.5 Flow of Setting Parameters

Procedure

1. Press the <Generate Template> button and enter the path and file name in order to generate a CSV file as a template.
2. Open this data using an editor and enter the signals into it.
3. Open the modified file in the I/O configurator and load the configuration in the RFM card by clicking on <Accept>.
4. Wait until the initialization phase is complete and the RFM card has been fed with these parameters.
5. Then press <Update Hardware> once again so that the signals defined are displayed in the resource tree.
6. Assign the generated signals to the virtual signal names.

11.5 ibaPADU-S-IT Platform

Local Peripherals of the ibaPADU-S system is available only if a device of type ibaPADU-S-IT has been selected as the platform.

ibaPADU-S-IT is the central unit for the ibaPADU-S family of modular devices for intelligent and decentralized inputs/outputs.

The modular concept is based on a module rack having a rear wall bus, in which the central unit and up to 4 other input/output modules can be plugged in.

There are modules available as I/O modules for analog and digital inputs/outputs for different signal levels, for current and voltage signals and for sampling rates of up to 1 kHz (buffered access) or max. 40 kHz (unbuffered access).



Hardware Documentation

Please refer to the ibaPADU-S-IT manual for detailed information on the ibaPADU-S properties (see "Support and contact, Page 330").

11.5.1 Settings

When you are connected with the platform and press the <Update Hardware> button, you see the following PADU-S settings.

PADU-S settings	Explanation
Interrupt source	DNS name of the PADU-S-IT, e.g. "S-IT-16-000074"
Module settings	depending on the I/O resource
Signal settings	depending on the I/O resource
I/O resources	PADU-S-IT incl. PADU-S module TCPIP_OUT GLOBALVAR

Module settings

In the "Module settings" section, all available modules of the ibaPADU-S system are displayed depending on the selected I/O resources.

Enabled:

You can completely disable specific modules.

Buffered access:

If the buffered access is enabled, further configurations in the ibaLogic program must be performed (see ibaPADU-S-IT-16 manual).

Only with a buffered access it is possible to reach a sampling rate of up to 20 kHz on the ibaPADU-S system.

In unbuffered access, max. 1 kHz is available (task interval = 1 ms)

Convert values into REAL:

If this option is selected, the signals are acquired not as INT, but as REAL and can be processed in the program without further conversion.



Note

Depending on the module type, the "Buffered access" and "Convert values into REAL" options are not available.

Signal settings

In the "Signal settings" section, all configurable signals of the particular I/O resource are displayed depending on the selected I/O resources and corresponding module settings (see ibaPADU-S-IT-16 manual).

11.6 TCP/IP Communication

The following types of TCP/IP communication are available:

- General TCP/IP connection via the module (for more information, please refer to section "TCPIP_SENDRECV, Page 104")
- ibaPDA data transmission using TCP/IP

TCP/IP communication, whose parameters can be configured in the I/O configurator, is presently limited to TCP/IP telegram transmission to the ibaPDA. In contrast to the native TCP/IP communication with the TCPIP_SENDRECV module, the module structure in the ibaPDA is supported here using a special protocol. On the WinXP or PADU-S-IT platforms, you can transmit a total of 16 telegrams each having 32 real values and 32 digital values to one or more ibaPDA receivers.



Note

Please note that the communication via TCP/IP is not real-time enabled. This means that data transmitted cyclically is not received in real time, and some telegrams may get lost etc.

11.6.1 TCP/IP Connection Settings

Select the appropriate channel from the tree on the left for the setting.

You can enable each of the 16 channels individually and enter the following parameters:

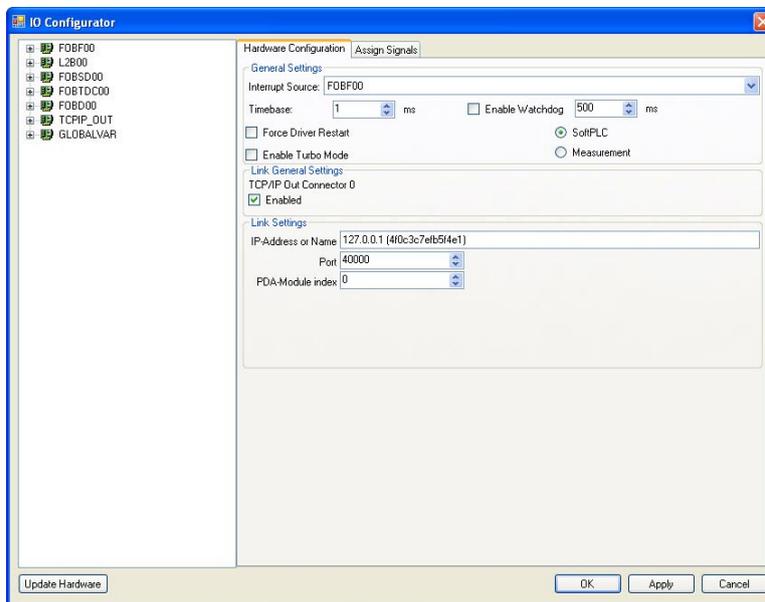


figure 113: TCP/IP Connection Settings

- IP address or name of the remote station, i. e. the computer on which the ibaPDA Server is running.
- Port number: This must match that in the ibaPDA. Default setting is 40000.

**Note**

On the ibaPDA system, the configured port must be enabled in the firewall to be able to measure the data with ibaPDA.

- PDA module number is a unique number 0..63. This must match a module index of the PDA system connected.

11.7 OPC Communication

The international OPC standard (OLE for Process Control) has prevailed for the connection to HMI systems (Human Machine Interface, control & monitoring) or for the measurement of slow signals.

11.7.1 OPC Server

The ibaLogic OPC Server provides all variables defined as "OPC visible" to the OPC Clients, which have been connected to the OPC Server. The OPC Server generally runs on the same machine as the ibaLogic Server and is connected with the PMAC via TCP/IP.

**Note**

The number of OPC variables permissible depends on the license purchased (Dongle).

**Note**

You can also run the OPC Server on another computer. You have to make a special inquiry with iba for this purpose.

An OPC Client finds the ibaLogic OPC Server under the name "**iba.Logic4OPC.1**". You can select the OPC variables using their variable names with the help of a browser service if the link is established.

The OPC Server works according to the DA (Data Access) specification V2.05a.

**Note**

You have to make a number of settings in DCOM and carry out safety guidelines for connecting the OPC Client to the OPC Server.

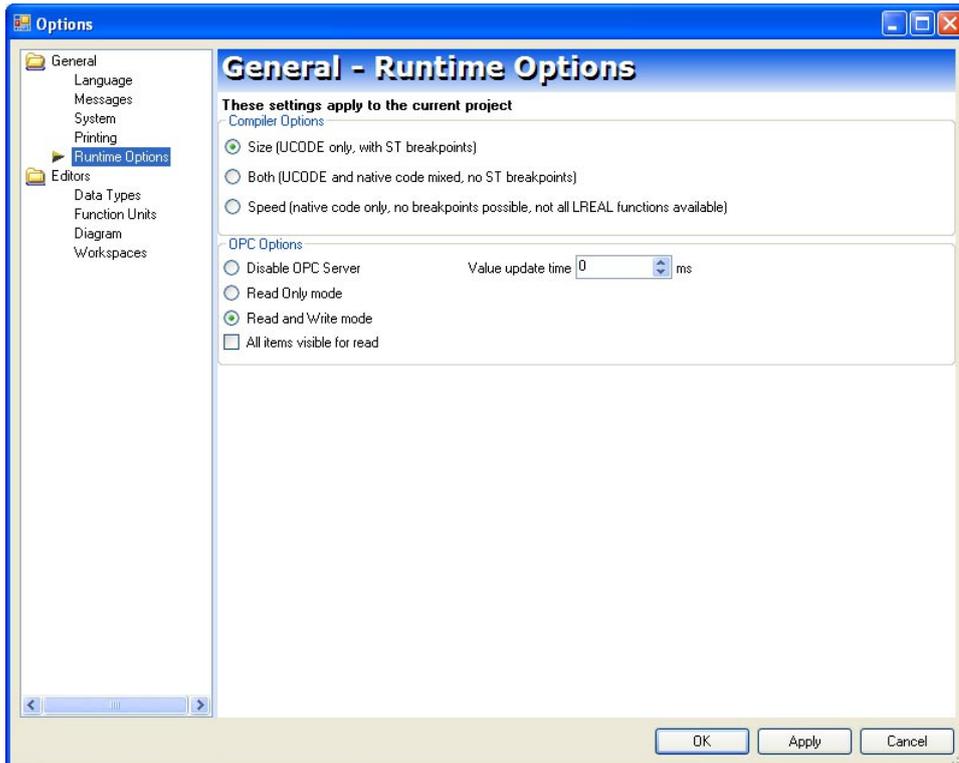
**Documentation**

There is a separate document for this purpose that you can get from iba on request.

1. In order to set certain OPC Server properties, open the options of ibaLogic by using the "Tools - Options..." menu item.



2. Switch to the tree on the left and select "Runtime Options".



3. Set the desired option.

Option	Explanation
Disable OPC Server	The OPC Server is shut down completely.
Read only mode	Even those variables that have been defined as "OPC write-enabled" can only be read.
Read and write mode	Default setting: Access is as defined in the variable parameters.
All elements are visible for reading	Even the variables that are not "OPC visible" can be read by the OPC Clients but cannot be written.

11.7.2 Setting the OPC Variable Parameters

An off-task connector (OTC) must be created for each OPC signal desired in ibaLogic. The OPC selection fields must be enabled in the "Off-Task Connector Edit" dialog box.

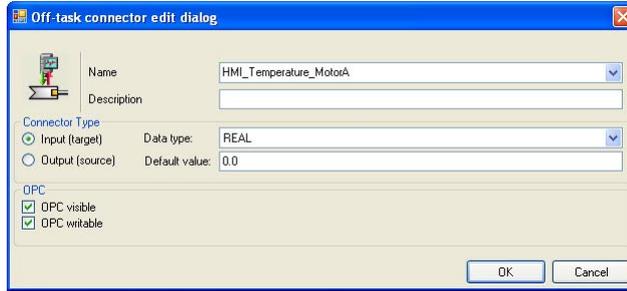


figure 114: Edit Off-Task Connector

The data types allowed for the OPC connection depends on the data type of the OPC Client used. Normally, you can use the elementary data types and arrays.

The shortest update time is 50 ms as seen from ibaLogic, however, you must note that this depends to a large extent on the data volume.

The OPC variables are marked and identified with special colors. For more information on setting parameters, see "Off-Task Connectors, Page 157".

You can find the OPC Server using the OPC Server browser under the name "iba.Logic4OPC.1".

12 Database Management

ibaLogic is a database-based application. In order to save intermediate results, you must backup the database regularly. ibaLogic-V4 provides support for this purpose by offering the options of automatic or manual backup.

12.1 Backup Database

You should always consider database backup before making comprehensive modifications to it.

12.1.1 Manual Database Backup

Backup always saves the complete and current ibaLogic database. This database may have several workspaces. You can see the workspaces that currently exist in the database in the client by selecting the "Open Workspace" menu option.

Prerequisite

- You have opened the ibaLogic Server dialog box.
- You have a connection to the database.

Procedure

1. Select the "Backup - Database..." menu.



The following dialog box opens:



- Choose a file name and a folder in which the backup file should be saved. Enable the option "Compress backup to zip", if the hardware configuration of the I/O manager and all DLLs present should also be saved.



- Click on <OK>.



Important Note

It is particularly recommended to backup the database before updating the ibaLogic version.

In the course of further advanced development of ibaLogic, during updates of ibaLogic, modifications are also made to the database with the help of database scripts. It is then no longer possible to revert to an older version.

12.1.2 Automatic Database Backup

You have to configure the setting in ibaLogic for automatic database backup.

Requirement

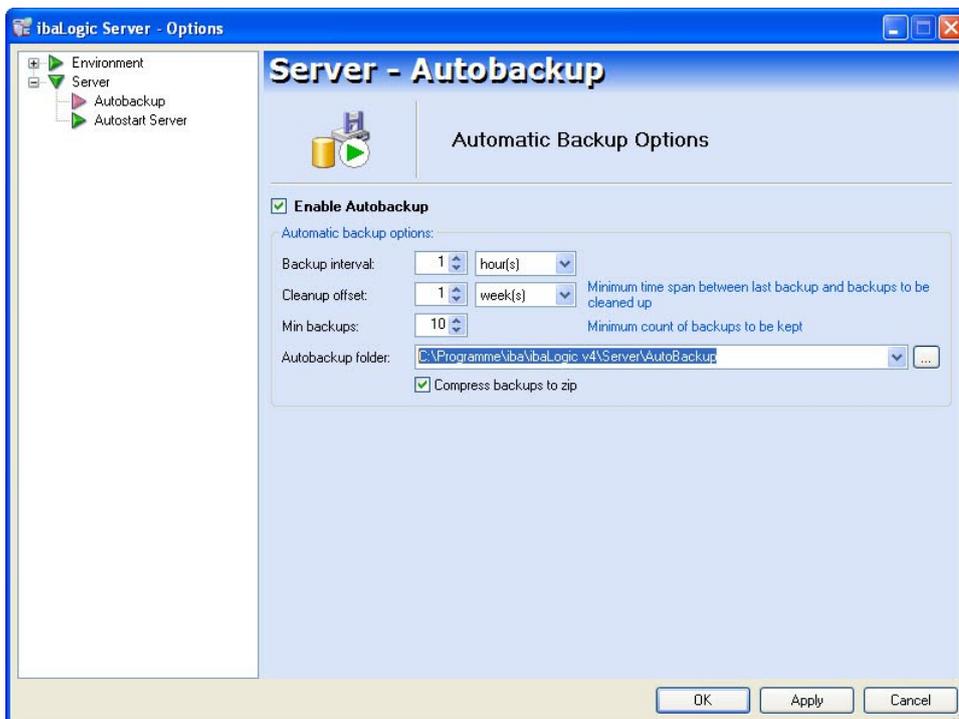
- You have opened the ibaLogic Server dialog box.
- You have a connection to the database.

Procedure

1. Select the "Tools - Options" menu.



2. Select "Server - Autobackup" in the folder tree.



3. Tick the checkbox "Enable Autobackup".
4. Select the time period for the backup interval.

Since modifications to ibaLogic projects can be made only with ibaLogic Clients, the interval is started only when a client is connected to the server. If a modification has been detected in the database, a new backup in the form of a *.bak or *.zip file is created after the interval time has elapsed.



Important Note

Specify different folders for automatic and manual backup respectively. Since the cleanup strategy cleans up only the folder specified for automatic backup, you can thus prevent your manual backup copies from being deleted.

The cleanup strategy is determined by the combination of the

- "Time until cleanup" (Cleanup offset)
- "Number of backup copies"

fields.

Option	Explanation
Backup interval	The setting creates a backup at the time interval specified.
Time until cleanup (Offset)	The setting creates backup copies until the minimum time span of the backup and cleanup offset is reached.
Number of backup copies	The option determines the minimum number of backup copies that remain at all times. The time period "Cleanup offset" is taken into consideration in the process.
Backup folder	The file names are assigned by ibaLogic: <i>"Autobackup_ibaLogic4_<Date, Time>.bak"</i> or <i>"......zip"</i> . Date and time are defined in YYYYMMDDHHMM format.
Compress backup to ZIP file	The option saves the backup as a ZIP file.

Example

If you have the settings as illustrated in the window given above, all backup copies that are older than one week are deleted. However, at least 10 files remain. These can be of any date.

12.2 Restore Database

Restore means that a previous version of the database with the workspaces contained in it will be loaded for editing.

Prerequisite

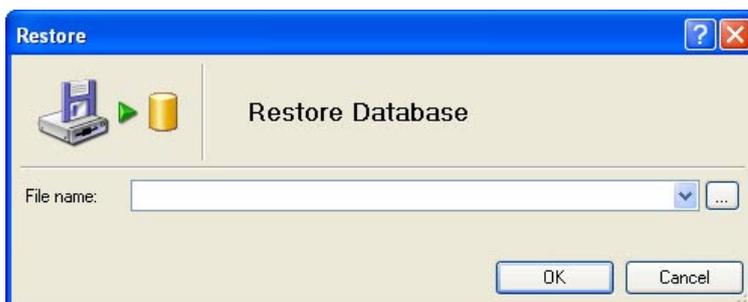
- You have opened the ibaLogic Server dialog box.
- You have a connection to the database.
- You have stopped the ibaLogic Server.

Procedure

1. Select the "Restore - Database..." menu.



The "Restore" dialog box is displayed.



2. Click on the <.....> button and select a backup database (ZIP or BAK file) from the folder that is open.

By default, ibaLogic provides the folder given below or notes the path to which the last access was made.



3. Click on the <OK> button to restore the backup copy.

The progress of the backup process is displayed on the screen. Thereafter, the server goes to the "Stopped" state.



4. If required, confirm any confirmation prompts that pop up.
5. Start the server for continuing the programming work via the client.

12.3 Reset Database

You can use this function to reset your current database to its original state (empty).



Important Note

This also deletes all data in the database.

First, make a backup copy of the database.

Prerequisite

- You have opened the ibaLogic Server dialog box.
- You have a connection to the database.
- You have stopped the ibaLogic Server.

Procedure

1. Select the "Databases – Reset Database....." menu.



2. Confirm the dialog with OK if you really want to reset the database.

13 Program Analysis, Debugging and Time behavior

You have various methods and tools available for program analysis and debugging.

You need to differentiate between whether the application is in the test environment or is already actively in use.

Description	Test environment	Active Use
Debugging Structured Text blocks using breakpoints	Yes	No (Program is stopped.)
Trace blocks or Log DLLs created by the user (e. g.LogFile_String_WriteDll.dll,)	Yes	Conditional (Time behavior)
Analysis of the Time behavior, curve shape etc. with ibaPDA Express	Yes	Yes
Writing DAT files for ibaAnalyzer with the help of the DAT_FILE_WRITE block	Yes	Yes

13.1 ibaPDA Express

The ibaPDA Express is used for checking a signal waveform quickly.

Requirement

The function is available only when the program has been switched online.

Procedure

- Start the ibaPDA Express by clicking with the mouse on the <ibaPDA Express> button in the toolbar.



Result

ibaPDA Express is opened with its own window within the ibaLogic application.

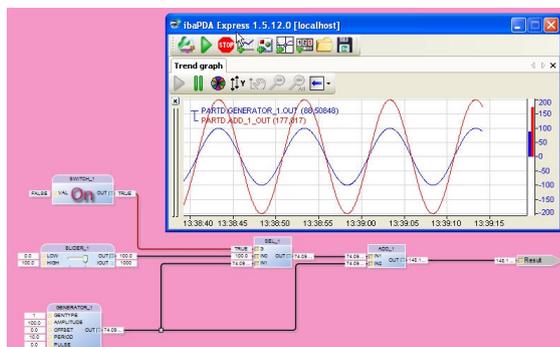


figure 115: ibaPDA Express with several signals

13.1.1 Controlling the Signal Display

The following toolbar is available for controlling the signal display.



figure 116: ibaPDA Express: Toolbar

The following table contains the explanation of the icons.

Icon	Name	Key operation	Explanation
	Start scrolling	<F6> (Switch)	Starts continuous display with the current time point. Active, when "Pause scroll" is pressed.
	Pause scroll	<F6> (Switch)	Stop the continuous display. After pressing this, a ruler appears in the graph that can be moved with the mouse and with which the curves can be measured. The signal values are displayed in the legend. You can move the X-axis using the mouse. In this manner, you can browse values from the past. Active, when the display is on.
	Assign signal colors automatically		All curves of this display are colored in accordance with the default scheme for each graph.
	Auto scale all	<F5>	All curves of this display are scaled automatically for each graph and the Y-axis.
	Restore manual scaling		Manual settings for scaling, where defined, are restored after auto scaling or zooming. Active, if manual scaling has been defined.
	Zoom out by one step	<F3>	Active only when the display has been zoomed. Return to the previous zoom factor (reduce).
	Zoom out all	<F4>	Active only when the display has been zoomed. Return to the initial (automatic) display.
	Scroll direction		You can change the scroll direction by selection in the pull-down menu.

13.1.2 Select Signals

You can drag signals by keeping the <Alt> button pressed from a connector and drop them into the ibaPDA Express window.

Optionally, you can:

- Display a signal in a separate signal strip.
To do this, drag the signal on the X-axis and a new strip is created.
- Place a signal in an existing strip.
To do this, drag the signal to the strip, and another Y-axis is created.
- Place a signal on an existing Y-axis (same scaling with one other signal).
To do this, drag the signal to this Y-axis.

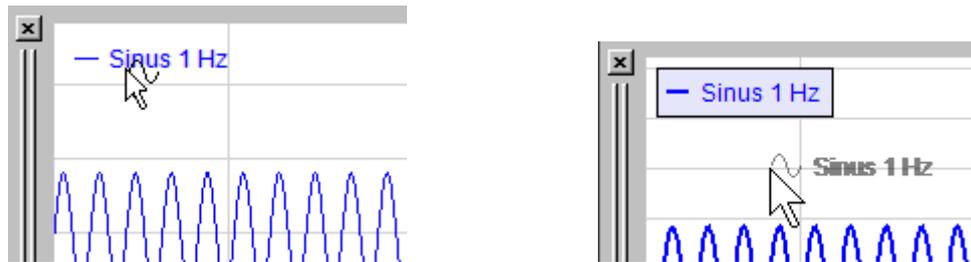
The new signals in one window are automatically assigned a new color. Those signal names having the same color are arranged in the top left section of the strip. Signals having a common axis are joined with a dash.

13.1.3 Move signal

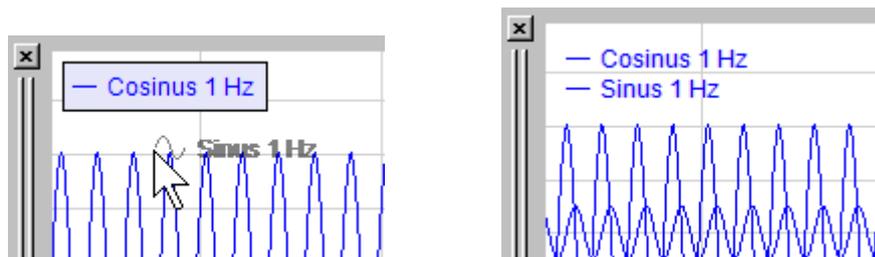
Signals can be moved between graphs and also beyond the limits of the window. This means that a signal can be dragged from one graph to another graph that already has a signal. You can differentiate between the signals with the help of the automatic color assignment.

Procedure

1. Move the mouse pointer to the name (Legend) of the signal that needs to be moved. The mouse pointer indicates with a wavy line that it has acquired the signal.



2. Drag the signal, keeping the mouse button pressed, to the other graph in order to drop it there in a free area.



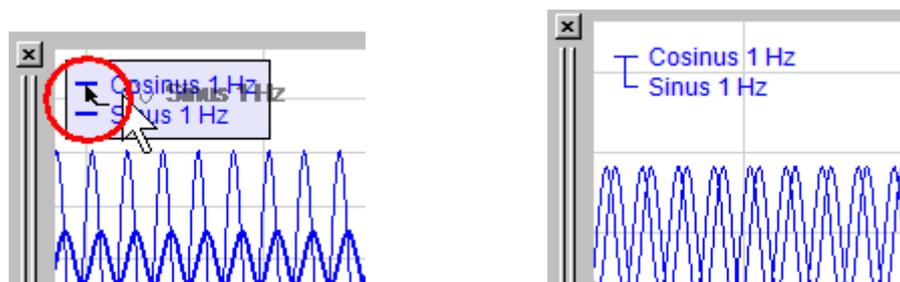
Result

You have created two signals with separate Y-axis.

Remark

Do not leave the signal in step2, but drag it to the existing signal until a small black arrow appears. In this manner, the same Y-axis is assigned to the signal.

In case of binary signals, you also determine the sequence of the signals. Binary signals are displayed below one another. Depending on whether the small black arrow docks above or – as illustrated below – the signal, the binary signal is displayed above or below it.



Result

You have created two signals with a common Y-axis.

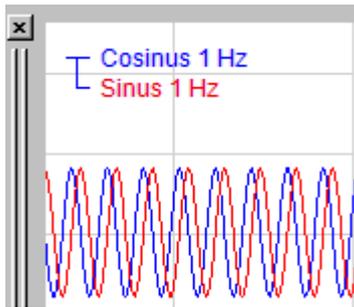
13.1.4 Mark the signals with color

You can mark the signals with colors in different ways:

- Automatically
- Manual setting

Procedure

- Press the <Assign signal colors automatically> button to assign colors to the signals automatically.



13.1.5 Remove Signal from the Display

Procedure

1. Place the mouse pointer in the graph on the name (Legend) of the signal that needs to be removed.
2. Click the right mouse button. The context menu is displayed.
3. Select "Remove signal".



Note

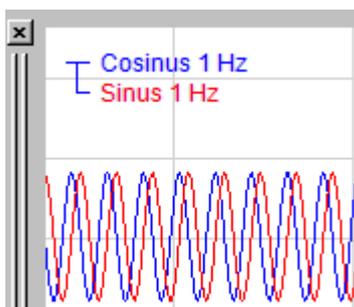
By removing the Y-axis, all signals are removed that are assigned to this axis.

13.1.6 Remove Graphs from the Display

There are different options to remove a graph.

Procedure

1. Click on the small cross at the top left above the top of the bar.



or

2. Click the right mouse button in a free area within the graph.
The context menu is displayed.
3. Select "Remove graph".

13.1.7 Scale Axes

13.1.7.1 Auto scaling

In order to display a signal over its entire amplitude range in one graph, it is recommended that you use the auto scaling feature. All signals or all Y-axes of the graph are scaled accordingly with respect to the largest amplitude.

Procedure

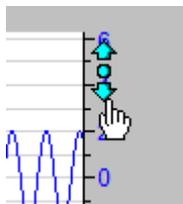
1. Press the right mouse button in the appropriate graph. The context menu is displayed.
2. Select "Auto scale".
3. If you would like to auto scale all graphs in one signal display, press the <F5> key or select the <Auto scale all> button.

13.1.7.2 Scaling with the mouse

You can change the scale of the signals in the Y-direction at the upper ends of the Y-axis scale using the mouse.

Procedure

1. Bring the mouse pointer close to the end of the scale until blue arrows appear.
2. Keep the mouse button pressed on the arrow pointing upwards: The scale gets expanded.
3. Keep the mouse button pressed on the arrow pointing downwards: The scale gets reduced.
4. Keep the mouse button pressed on the dot between the arrows: Auto scaling is carried out.



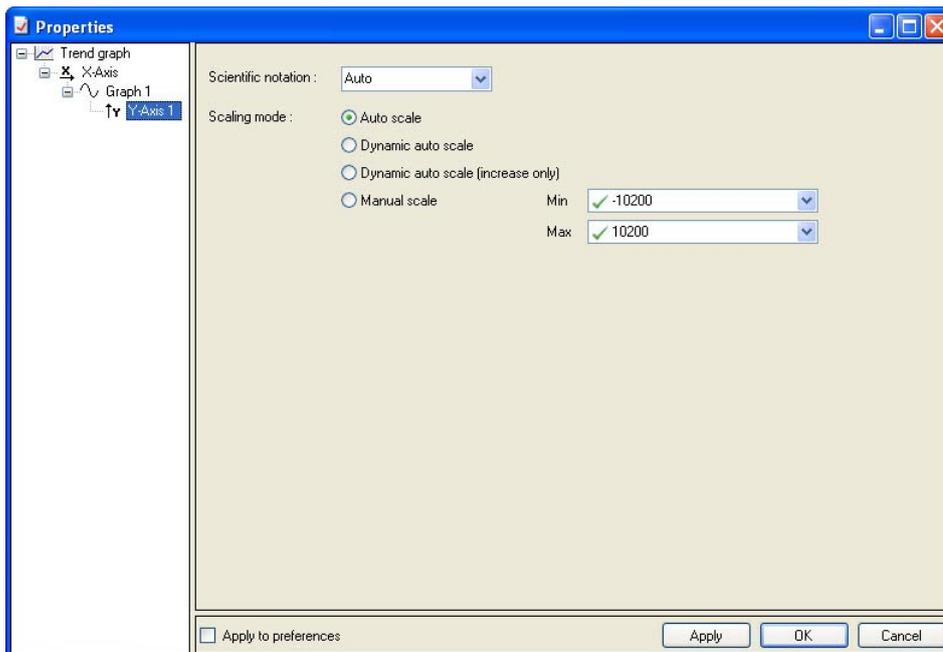
Tip

If you are using a mouse with a scroll wheel, you only need to position the mouse pointer on the scale. You can change the scale using the scroll wheel. This functionality is also available on the X-axis.

13.1.7.3 Scaling using the display settings

Procedure

1. Click the right mouse button in the area within the desired graph.
2. Select "Properties".
The "Properties" dialog box is displayed.
You can specify manual scaling using the "Y-axis" option. If a graph has multiple Y-axes, there is a separate tab in the dialog box for each Y-axis.
3. Accept the settings with <Apply>.



Result

All signals that are assigned to the corresponding Y-axis are scaled with the same setting.

Remarks

By selecting the "Apply to preferences" option, you select the settings configured as the default settings.

13.1.8 Move Scales

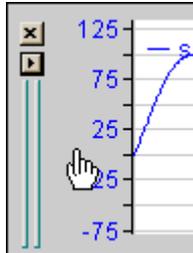
You can move the X-axis as desired in the pause mode of the display.

Requirement

Y-axis: Auto scaling is not selected.

Procedure

1. Position the mouse pointer on the Y-axis until the hand icon appears.
2. Press the left mouse button in order to move the scale upwards / downwards or to the left / right.

**13.1.9 Zoom Function**

The zoom function affects both the X and Y directions.

If you zoom in a graph, all other graphs located in the same display also get zoomed. A signal display can always maintain only one time base for all the graphs that it contains.

When the display is active and running, zooming expands the time base and hence enlarges the display. The signals run through faster, since the same geometric length of the X-axis is converted to fewer units of time.

Zooming in general

1. Press the <Shift> button and zoom simultaneously with the mouse. Only the X-axis is zoomed.
2. You can restore the original and un-zoomed display using the  button or the <F4> key.

13.1.9.1 Zooming in (Enlarge)

You can zoom in all over in one strip. In the zoomed in state, you can change the scale in the Y-direction without affecting the zoomed section of the X-axis.

Auto scaling in the Y-direction pertains to the values in the zoomed (= visible) area.

Requirement

You have zoomed out.

Procedure

1. Draw a rectangle using the left mouse button so that the area selected is enclosed.
2. Release the mouse button.

13.1.9.2 Zoom out (Reduce)**Requirement**

You have zoomed in.

Procedure

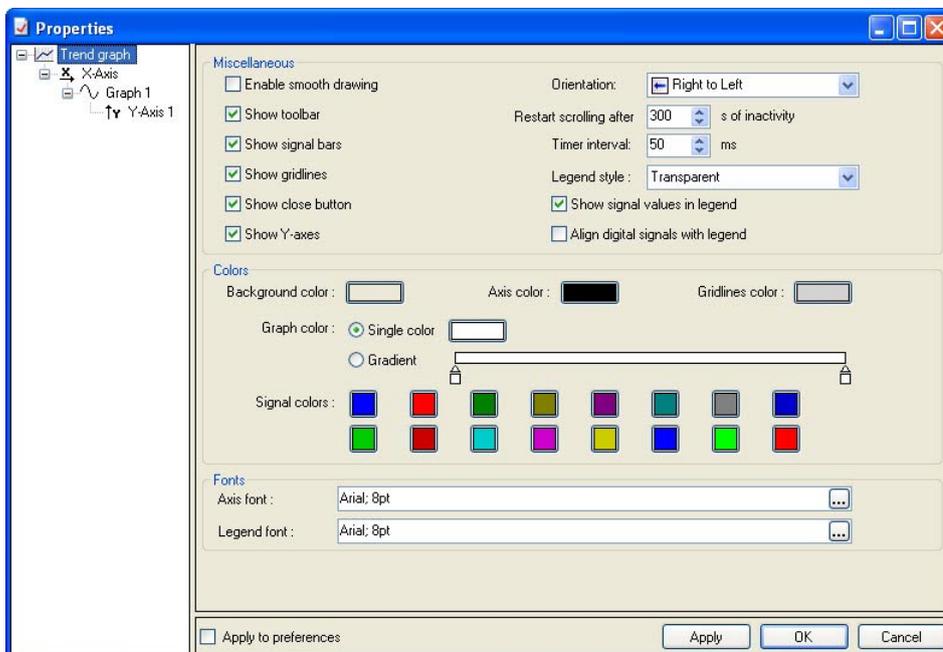
1. Press the <Zoom out one step> button or the <F3> key to achieve reduction step by step.
Thus, with each action, all previous zoom steps are reversed one after another.
2. You can restore the original and un-zoomed display using the <Zoom out all steps>button or the <F4>key.

13.1.10 Trend graph Properties

You can configure general settings for the display of graphs in the Trend graph Properties dialog box.

Procedure

1. Click with the right mouse button on the signal strip.
2. Click with the left mouse button on "Properties...".
3. Select "Trend graph" in the structure on the left.



13.1.10.1 Miscellaneous

Option	Explanation
Activate smoothed display	This option smoothens the graph lines as they are displayed.
Display toolbar	This option displays the toolbar.
Displays signal values in the legend	This option displays the signal values in the legend.
Display bars	This option displays the bar associated with the graph.
Transparent legend	This option makes the legend style transparent.
Scroll direction	The scroll direction is set.
Restart scrolling	Configuration of a certain time in seconds, which restarts scrolling after inactivity for this period of time.
Refresh interval	Setting for the time intervals at which the display should be refreshed.
Align digital signals with the legend	This option aligns the digital signals with the legend.

13.1.10.2 Colors

You can use this dialog screen to change the color scheme for the trend graph display and the pen colors for the curves.

- Click on the respective color button to change the color. Select the desired color from a color palette.
- Background, axes, gridlines:
 - Click on the respective color button to change the color. Select the desired color from the color palette.
- Graph:
 - Background color in the signal strip uniform or with progressive color.
 - Double click on the small box at the end of the color bar and select the color from the color palette.
If required, you can double click on the color chart to add other color tabs and to color them, and these can also be moved. In order to delete a color tab, mark it with a mouse click (black arrow tip) and press the key.
- Signals:
 - You can use these pen colors to define 16 curve colors that are available for the trend curve display. The program assigns colors to the trend curves automatically based on these 16 colors. The pen colors are also provided in the signal definition in the signal grid in the sequence shown here (line wise from the top to the bottom).

13.1.10.3 Fonts

The fonts are defined for the lettering of the axes and the legends (Signal names). You can open the dialog box to change the font using the <...>browser button at the end of the line.

13.1.10.4 Signals

If you call up this dialog box in a graph or for an existing trend graph, in which signals are being displayed, the signals with their current setting are listed, including the colors.

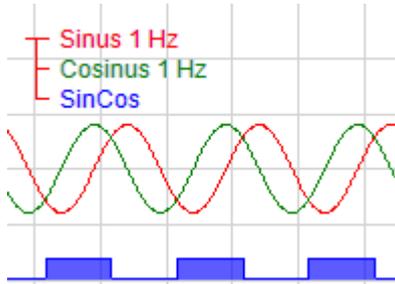


figure 117: Trend graph

Graph: 0	Offset	Color	Filled	Pen width
IO_Kon.ibaLogicFB1_1.Sinus	0		<input type="checkbox"/>	1
IO_Kon.ibaLogicFB1_1.SinCos	0		<input checked="" type="checkbox"/>	1
IO_Kon.ibaLogicFB1_1.Cosinus	0		<input type="checkbox"/>	1

figure 118: Graphical signal settings

You can choose the color for each signal from a selection list in the cells of the "Color" column of the table.

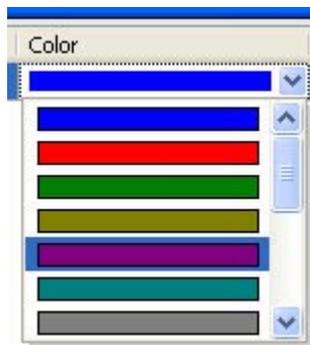


figure 119: Color selection list

13.1.10.5 X-axis

Time range

You can specify a fixed time range in seconds instead of automatic scaling, and this is shown in the display. In this manner, you control the speed and the expansion of the signal in the X-direction in the display.

Time range : s fixed axis

figure 120: X-axis: Properties

Fixed axis

Normally, the time axis moves with the signal so that new values sampled are always shown at the border of the graph in the display. Using the "Fixed axis" option, the time axis from the current time point for the period (time range) configured is fixed and the sampled values are written into the empty graph. If the graph is filled, the next (empty) time range is displayed and sampled values continue to be written.

13.1.10.6 Y-axis

If you have created more than one Y-axis in a graph, the settings dialog screen has multiple "Y-axis #" tabs. Thus, you can configure settings for all Y-axes separately.

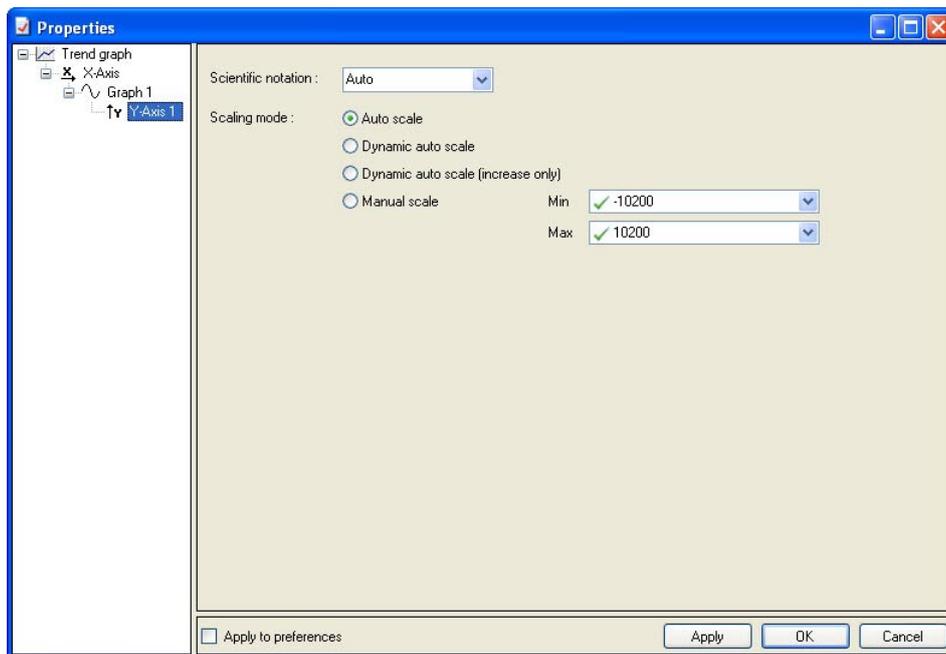


figure 121: ibaPDA Express: Display settings

13.1.10.7 Scientific notation

- "Auto"
- "Always"
- "Never"

Option	Explanation
Auto	Depending on the size of the scale values (number of places before or after the decimal point), the scales are labeled in scientific notation (power of 10) or not.
Always	Scale values as power of 10
Never	Scale values always with digits before and after the decimal point

13.1.10.8 Scaling mode

- "Auto scale"
- "Dynamic auto scale"
- "Dynamic auto scale" (increase only)
- "Manual scale"

Option	Explanation
Auto scale	Default setting; when displaying one or more graphs, the Y-axis of the strip is scaled once in accordance with the lowest and highest of all values occurring (when involving a signal).
Dynamic auto scale	When you enable this option, the scaling is continuously adjusted with the highest signal amplitudes. If the amplitudes go beyond the signal strip again, the scaling is further reduced.
Dynamic auto scale (increase only)	When you enable this option, the scaling is continuously adjusted with the highest signal amplitudes. If the amplitudes go beyond the signal strip again, the scaling remains unchanged.
Manual scale	You can specify the starting (Min.) and end (Max.) value of the scale manually when you select this option. (Visible only when the dialog box is opened from the context menu in the signal strip, and not with the presets.)

13.1.11 Extended Functionality

You can enable the extended functionality using the icon in the title bar from the context menu.

The following functions are available:

- "Toolbar"
- "Signal Tree"
- "Fullscreen-View"

The display of the ibaPDA Express gets extended when you select the menu.

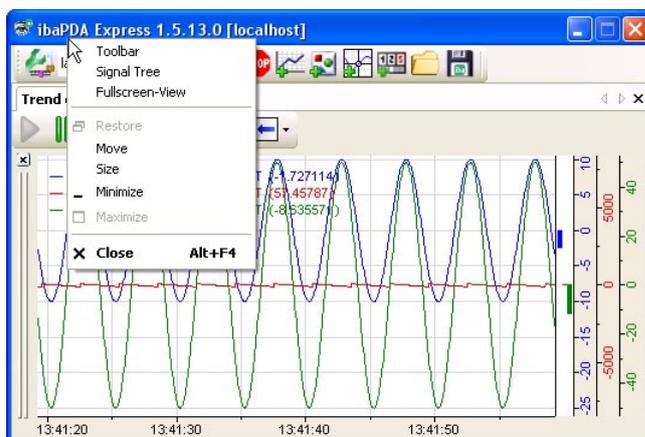


figure 122: Context menu "ibaPDA Express"

Toolbar



It displays a toolbar having the following elements:

Icon	Name	Explanation
	Modify presets	Opens the properties menu.
	Real time	Start / Stop function of all strips.
	Add trend graph	Add another trend graph The trend graphs can be arranged as desired by holding the "Trend graph" bar with the left mouse button and moving it. The docking points become visible.
	Add QPanel Add scope view Add digital meter	Add supplementary functions QPanel, scope view, digital meter to Express.
	Load view	Open an ibaPDA Express configuration as an XML file.
	Save view	Save an ibaPDA Express configuration as an XML file.

Signal tree

Displays a tree view of all variables contained in the program. These can be placed in a trend graph using Drag & Drop or by double clicking.

Full screen view

ibaPDA Express is displayed in full screen mode. You can exit this mode by pressing the <F10> key.



Other Documentation

You are requested to refer to the appropriate add-on documentation of the ibaPDA system for the description.

13.2 Time behavior

Depending on the platform, ibaLogic provides a deterministic Time behavior (Real-Time behavior).

Platforms:

- ❑ Windows XP/7:
non-deterministic, relatively stable cycle times for task times of ≥ 5 ms.
- ❑ PADU-S-IT:
deterministic, very stable cycle time for task times of ≥ 1 ms.

The tasks of ibaLogic have a base time slot of minimum 1 ms, and this is based on the interrupt time base, which can be configured under the "Tools I/O configurator" menu. This is the minimum task interval.

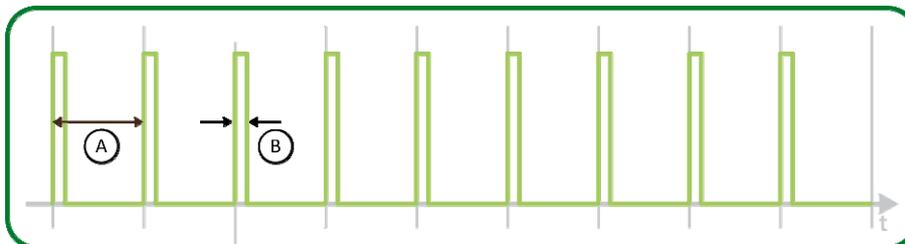
You cannot have faster tasks. It is possible that certain iba modules sample data at 50 μ s and forward these as an array of values (Packets) to ibaLogic. ibaLogic then processes the values in the secondary clock cycle. Further information, please refer to „Buffered Mode, Page 192“.

For the purpose of task handling, ibaLogic logic checks the tasks pending for processing at the basic clock cycle configured, and enters these in an internal task list. This task list is evaluated cyclically from the top to the bottom and tasks evaluated are removed from the list.

It must also be noted that the basic clock cycle configured is also the clock cycle in which inputs can be read and outputs can be written.

IbaLogic knows only the interval task.

The interval task is started in accordance with the time interval configured. The program linked with it has its own evaluation time.



A Interval

B Evaluation time

figure 123: Interval task

13.2.1 Evaluation time

The program evaluation times of various programs in the entire user project is also important for the consideration of the Time behavior.

ibaLogic provides the evaluation time for various interval tasks to the user to check the system loading as a number and as a bar.

Evaluation time: 34,36% avg. 0,3436 ms 

In this example, considering a 1 ms interval task, the percentage value means that 34.36 % of 1 ms is required. This means that this value is a percentage of the time slot configured for the task. 34.36 % for a task time slot of 1 ms works out to 0.3436 ms of CPU time for the program.

13.2.2 Turbo mode

In order to prevent ibaLogic from getting temporarily blocked by Windows, you can assign one processor core to ibaLogic exclusively in multi-core systems.



Note

In order to ensure flow and performance as deterministic as possible, iba recommends:

- For task times < 20 ms:
Use an iba interrupt source
(ibaFOB card or similar)
- For task times < 5 ms:
Use the turbo mode



Tip

The Time behavior can also be controlled with the help of compiler options using the "Tools - Options - Runtime options" menu.

- "Size:"
Default setting, interpreter mode (UCODE)
with ST breakpoints possible
- "Both:"
Interpreter mode and native code mixed,
no ST breakpoints possible
- "Speed:"
only native code, no breakpoints, and not all LREAL functions are
available (e. g. all exponential functions)

In addition, ibaLogic differentiates between the "Measurement" mode and the "Soft PLC" mode. Settings see "General Settings, Page 180".

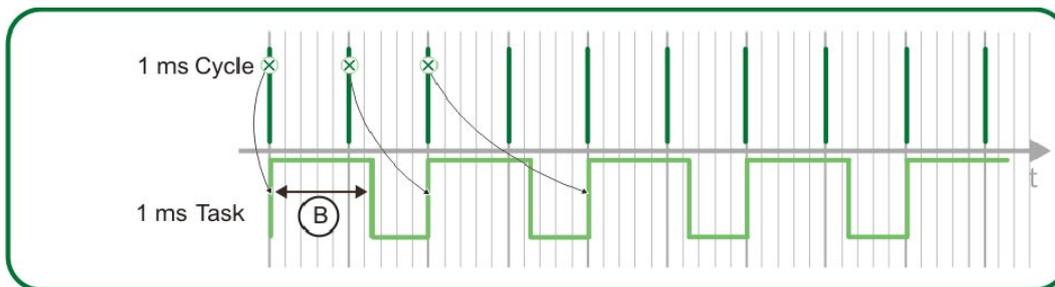
13.2.3 Messung

This operating mode ensures that ibaLogic does not lose any input sample. This is also true when individual tasks within ibaLogic need to be suspended. The runtime system of ibaLogic ensures that the data are made available equidistantly in the task interval configured. If tasks get suspended, the system makes up for the cycles. As a result, with task suspension for limited time, it may happen that ibaLogic, at times, evaluates only those values that belong to the "past".

Nonetheless, it is always ensured that, for example, values that are equidistant and correct are available for FFT analyses. Permanent suspension or blockage leads to buffer overflow. Such customization is not acceptable.

You must make considerations regarding the modes of operation possible for reading in the hardware signal inputs.

In the "Measurement" mode, the hardware input signal status is buffered in accordance with the task interval configured. The program then works with the oldest buffered value when it starts next. This means that the "Soft-SPS" mode and the "Measurement" mode work the same way when the program processing times < the interval time. If the processing times are greater, you have buffer overflow of the sample values in the "Measurement" mode.



B Evaluation time

figure 124: Buffer overflow – Shifts

Example: The dark green 1 ms clock cycle saves the value that is processed when the task begins (light green). The black arrow indicates the sampled value with which the task works and how the buffer overflow condition develops. The evaluation time of a task is more than 1 ms, and hence, there is a time shift.

13.2.4 Soft PLC

In this operating mode, which is suitable for control and regulation tasks, ibaLogic ensures that only the latest signal states are processed. In contrast to the "Measurement" operating mode, it does not matter here whether samples get lost or not. On the other hand, it is desired that current data, as far as possible, that is, data from the latest I/O transfer cycle is available.

Data is read in from the input resources with every cycle before executing the first task. The aging time of the resources is determined by the base cycle of ibaLogic that is configurable. If, for example, this base cycle is 10 ms and the first task is configured with a cycle of 50 ms, this task finds input data that is definitely not older than 10 ms. The data can, however, be newer.

Output values are written in both modes by each task in the cycle at the end of the required task evaluation time, provided output resources have been included in the plan.

In the "Soft PLC" mode, the current hardware input signal status is read in and processed at the start of the task.

13.2.5 Time considerations with multiple tasks

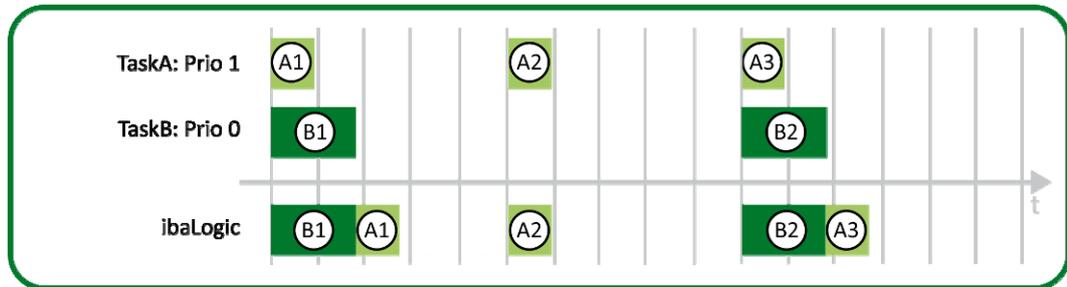


figure 125: Evaluation without overflow – 2 tasks having different interval time and priority

The 2 rows above represent the individual tasks in the theoretical evaluation sequence if they were to run independently. The numbers are a counter for triggering the tasks (1st trigger, 2nd trigger...).

An interval task A having an interval time of 5 ms is displayed in the uppermost row. The priority is 1, i.e. of lower priority compared to the 10ms task B having priority0 (second row). The width of the bar (impulse) is equivalent to the evaluation time of the task of the associated program. The background represents a clock cycle grid. The impulse always begins at the interval time set.

Practically, however, the tasks are executed "serially". This is illustrated by the lowermost row. At the starting time point, ibaLogic sees the tasks that need to be evaluated, and evaluates them one after another in accordance with the priority entered. First, task B, since it has the higher priority, and after its evaluation time, the task A...

To clarify the actual situation, the program evaluation times shown are taken to be very large. In reality, the evaluation times are primarily of the order of μ s, so that, for example, 20 tasks can be evaluated in 5 ms without a problem (empirical value).

13.2.6 Worst-case considerations

If you assume a longer evaluation time for task A and task B, suspension or time shift is generated. Suspension or time shift means that the task is no longer started at the expected time point, since another program is still being evaluated. The task with the higher priority is started at the correct point in time.

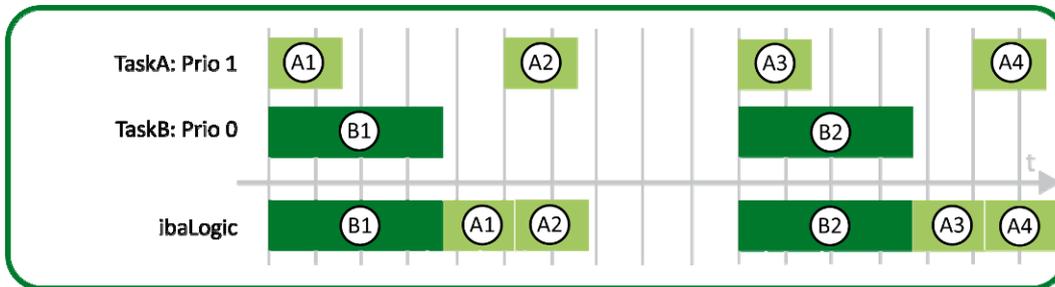


figure 126: Task evaluation with time shift (suspension)

The evaluation times have been selected in such a manner that both tasks together require more than 5 ms, and hence, task A cannot be started at the exact time interval foreseen. If a base cycle of 1 ms has been configured, a check is conducted at each cycle to see whether a task needs to be triggered. In the example here, task A (5 ms, priority 1) and the task B (10 ms, priority 0). These are entered in an internal list according to their priority and then started.

13.2.7 Explanation of the case above

The jobs of the internal list are illustrated in the figure "Task evaluation with time shift - Excerpt".

At the outset, ibaLogic sees that task A (5 ms, priority1) and task B (10 ms, priority 0) need to be executed and enters them in the internal job list according to their priority. Tasks that have been started are removed from the job list, new ones are added, and this is how the above figure emerges.

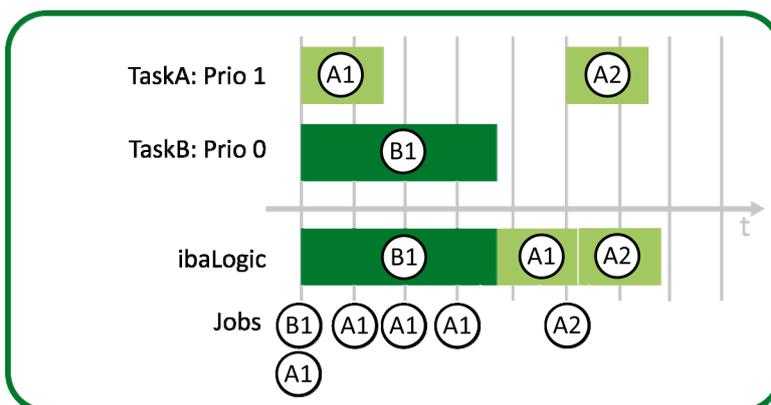


figure 127: Task evaluation with time shift – Excerpt

13.2.8 Task evaluation with time shift

Let us assume that task A has been configured with an interval time of 2ms (with the same program evaluation times), in which case, certain **cycles are lost**.

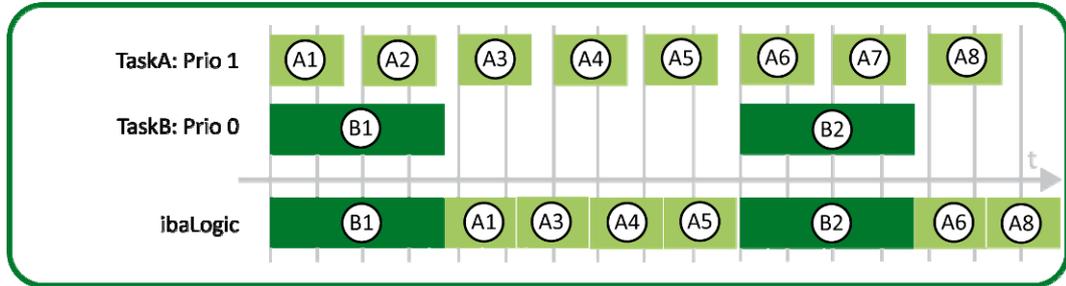


figure 128: Task evaluation with time shift

A different picture emerges if the priorities are interchanged.

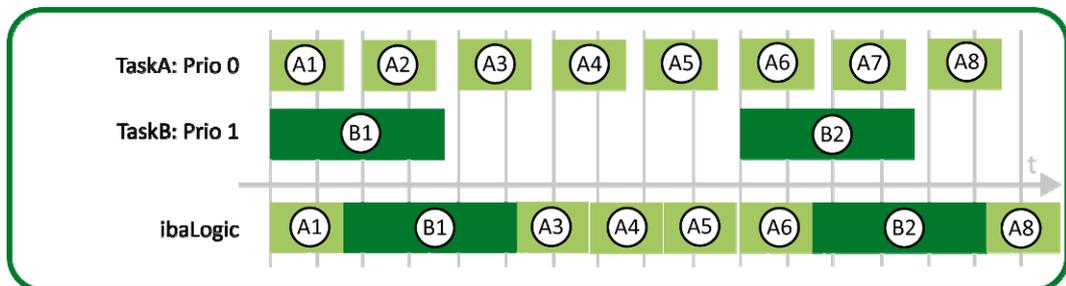


figure 129: Task evaluation with time shift (reversed priority)

Another consideration is the "Soft PLC" mode and the "Measurement" mode.

In the "Soft PLC" mode, the hardware inputs are always read at the beginning of the task (x point in the following figure).



figure 130: Hardware inputs in the "Soft PLC" mode

The situation in the "Measurement" mode is different.

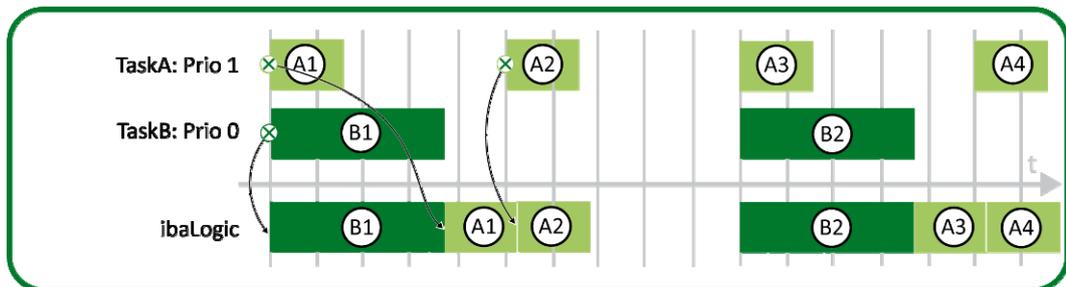


figure 131: Hardware inputs in the "Measurement" mode

Here, the input signals are buffered in time, but are evaluated with a delay in case of a time shift or task suspension.



Important Note

In general, there is a time shift or task suspension if the sum of the program evaluation times exceeds the smallest interval time used. There is buffer overflow if this time shift is permanent. The programs and the computer no longer work in line with the requirements. In case of temporary time shift or task suspension, it depends on the respective application whether this can be tolerated.

Data is written to the hardware outputs in the next base clock cycle after the evaluation time has ended. Hence, it may be meaningful to configure the base clock cycle to be faster than the task interval. If, for example, the evaluation time is 50 μs , the task interval time is 5 ms and the base clock cycle is 1 ms, data is written to the outputs after 1 ms.

13.3 Debugging

The following errors may occur:

- Program errors
- Compilation errors

13.3.1 Program errors

Frequently occurring errors in programs:

- Errors in user-defined function blocks
- Division by 0
- Incorrect signal trends
- Incorrect evaluation order or sequence

13.3.1.1 Errors in user-defined function blocks

In order to trap logical errors, ibaLogic-V4 provides you the option of setting so-called "Breakpoints" in the function blocks, so that you can check the execution of your ST code. For more information, please refer to "Structured Text Editor, Page 126".

13.3.1.2 Division by 0

If the following message appears in the event window, it means that division by 0 has occurred.

```
"Exception: OnlineServer: PMAC Status: Division by Zero in  
program.functionblock, Offset 0x0022, Stack 0x0001"
```

The message indicates the location at which the division by 0 has occurred. In the example given above, the error has occurred in the function block "*functionblock*" in the program "*program*".

13.3.1.3 Incorrect signal trends

In order to be able to check evaluated values, ibaLogic-V4 provides the tool ibaPDAExpress. You can display the signal trends in real time with the help of this tool and thus, track whether your block is yielding the expected output values with various input parameters.

13.3.1.4 Evaluation sequence

If, in spite of error-free function blocks and macro blocks, the evaluation does not run as you expect it to, it is possible that the problem lies with the evaluation sequence.

In order to check the blocks that are evaluated first, you can view the evaluation sequence of the corresponding program and thus, unearth any errors in the sequence.

For more information, please refer to "Evaluation sequence, Page 60".

If your program contains feedback paths, it is necessary to know the block that is in the first or last position in the evaluation sequence.

13.3.2 Compilation errors

Although the syntax of the ST in the user-defined FBs is checked by the block generator prior to compilation, it may happen under certain circumstances, that compilation of the IL code generated fails.

In such a case, you receive an error message in the event window that indicates this.

If there is a message in your event window that appears as follows, please scroll up using the slider on the right border until you can see the first error message.

Example:

```

1 [01.03.2010 14:19:14] [<Computername>] [ibaLogicClient] Info:
2 Generation started...
3 [01.03.2010 14:19:14] [<Computername>] [ibaLogicClient] Info:
4 Compilation started...
5 [01.03.2010 14:19:14] [<Computername>] [ibaLogicServer] Info: TIMER
6 Generation: Ticks since start of IL generation: 31, that is 0,03
  seconds
7 [01.03.2010 14:19:15] [<Computername>] [ibaLogicClient] Exception:
8 IL compilation failed: Compilation ended with errors.
9 [01.03.2010 14:19:15] [<Computername>] [ibaLogicServer] Info:
10 Building resource C:\Documents and
11 Settings\<Benutzername>\Application
  Data\ibaLogic\NewWorkspace\NewProject\${ENV$}\Resource\Resource.MAK.
12 C:\DOCUMENTS AND SETTINGS\<Benutzername>\APPLICATION
13 DATA\IBALOGIC\NEWWORKSPACE\NEWPROJECT\CustomTypes.typ
14 C:\DOCUMENTS AND SETTINGS\<Benutzername>\APPLICATION
15 DATA\IBALOGIC\NEWWORKSPACE\NEWPROJECT\FB_STRINGOUT.POE(3,5,2): E:
16 S3023: Invalid operand type for this operation.
17 1 error(s), 0 warning(s) -
18 C:\DOCUMENTS AND SETTINGS\<Benutzername>\APPLICATION
19 DATA\IBALOGIC\NEWWORKSPACE\NEWPROJECT\FB_STRINGOUT.POE.
20 C:\Documents and Settings\<Benutzername>\Application
21 Data\ibaLogic\NewWorkspace\NewProject\T00_INPUT.POE(2,9,14): E:
22 S3026: Undeclared identifier.
23 C:\Documents and Settings\<Benutzername>\Application
24 Data\ibaLogic\NewWorkspace\NewProject\T00_INPUT.POE(3,2,6): E:
25 S3005: This is not a function block instance.
26 3 error(s), 0 warning(s) - C:\Documents and
27 Settings\<Benutzername>\Application
28 Data\ibaLogic\NewWorkspace\NewProject\T00_INPUT.POE.
29 3 error(s), 0 warning(s).

```

If compilation fails, always begin with the first error message that appears in the event window.

In order to be able to find the errors, please proceed as follows:

- ❑ Enlarge the event window so that the events starting from "Compilation started" are displayed.
- ❑ Look for the first message that contains an error. The other messages are possibly errors as a consequence of the first error. In the example given above, it is the message

```
"C:\DOCUMENTS AND SETTINGS\\APPLICATION
DATA\IBALOGIC\NEWWORKSPACE
\NEWPROJECT\FB_STRINGOUT.POE(3,5,2): E: S3023: Invalid
operand type for this operation"
```
- ❑ Copy the path in which the erroneous block is located to the clipboard and insert it in the address line of Explorer. In the example given above, this is "C:\DOCUMENTS AND SETTINGS\\APPLICATION DATA\IBALOGIC\NEWWORKSPACE\NEWPROJECT"
- ❑ You will find the erroneous block there, in our case "FB_STRINGOUT.POE". Open it using an ASCII editor that displays the line numbers, e. g. NotePad++. You see, for example, the following program code (Example given above)

```

1 FUNCTION_BLOCK FB_StringOUT
2     VAR_INPUT
3         i1 : INT;
4     END_VAR
5
6     VAR_OUTPUT
7         o1 : IBA_STRING;
8     END_VAR
9
10 (** o1 := 'TestString' + int_to_string(i1); **)
11     (* assign - Stmt *)
12     LD 'TestString'
13     ADD ( i1
14         int_to_string
15     )
16     ST o1
17
18 END FUNCTION_BLOCK

```

- ❑ At the end of the block, you see a set of three numbers, e. g. (3,5,2). This indicates the following:

1st number: Region in which the error has occurred.

1 = Program name / FB name/Function name (Line 1 above)

2 = Variables range, begins with "VAR" (Line 2 above)

3 = Program, begins after the last END_VAR (Line 9 above)

2nd number: Line number within the section (5 is equivalent to line 13)

3rd number: Column number (or tab number) in the line (2) concerned

- ❑ the erroneous line is thus "ADD (i1 ... ". Search the last comment line above this statement. In this, you can see the source text of the ST statement

```
o1 := 'TestString' + int_to_string(i1);
```

- ❑ The error is the following: The ADD operand is not permissible for strings.
Possible cause: in other compilers, e. g. in ibaLogic-V3, you can combine strings with '+'. The ibaLogic-V4 compiler always interprets '+' as addition. In order to append strings to one another, please use the CONCAT function.
- ❑ The correct statement for ibaLogic-V4 is:

```
1 v1 := int_to_string(i1);  
2 o1 := concat('TestString',v1);
```

Sometimes you can get more information from the "CompilerOut.txt" file, which helps you to eliminate the error.

The "CompilerOut.txt" file is located in the following folder in Windows systems:

- ❑ German system
C:\Dokumente und Einstellungen\<<Benutzername>\Anwendungsdaten\
 \ibaLogic\<<Workspacename>\<Projektname>\\$GEN\$\
- ❑ English system
C:\Documents and Settings\<<Username>\Application
 Data\ibaLogic \<<Workspacename>\<Projektname>\\$GEN\$\

13.4 Performance Limits

ibaLogic-V4 has been developed for the 32 bit variant of Windows and has certain limitations on account of its architecture:

- ❑ Maximum RAM size: 4 GB
- ❑ Maximum process size (i. e. memory that a runtime system can occupy)
 - For WinXP platform: 2 GB
 - for the PADU-S-IT platform: 32 MB

The **Microsoft SQL Server 2005 Express** used by ibaLogic-V4 has the following system-bound performance limits:

- ❑ Maximum database size: 4 GB
- ❑ Maximum 16 instances on the same machine
- ❑ Support for only 1CPU and 1GB RAM

Moreover, there are limitations resulting from the compiler used that is integrated in ibaLogic.

This has a maximum segment size of about 64 kB. This means that you cannot define as many variables as you please within a block (function block or macro block). Nonetheless, if you exceed the permissible limit of 65292 bytes, an error message is output.

13.4.1 Example

You have an array of 8,158 LREAL elements, and each of them occupies 8 bytes, which means that you occupy 65,264 bytes with this array in a function block, plus the array header of 12 bytes, that is, 65,276 bytes.

You can provide the function block with only one input and one output that occupies only 4 bytes, so that you do not exceed the set limit. Since the header of the FB also occupies another 8 bytes.

Element	Bytes
FB Header	8 Bytes
Array Header	12 Bytes
Array [0 ... 8157] LREAL	8158 * 8 Bytes = 65264 Bytes
Input i1 (DINT)	4 Bytes
Output o1 (DINT)	4 Bytes
Total	65292 Bytes

The remaining 244 bytes are required by the compiler for administrative information. The following figure illustrates the structure of the data segment in a simplified form.

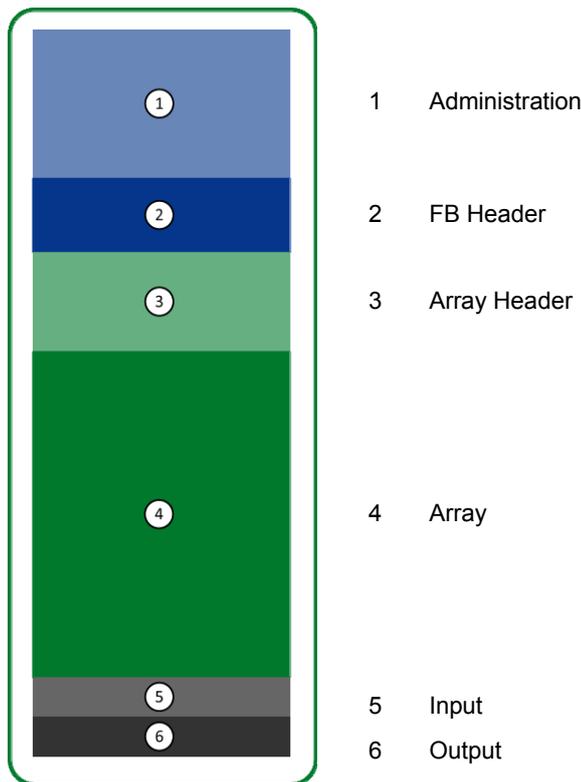


figure 132: Data segment(64KB)

You can create up to 600 programs / tasks within a project, but this is rather a theoretical limit considering grounds of clarity.

The number of projects within a workspace is limited only by the maximum database size on the server.

If you use another SQL server, you can learn about the size specification from your system administrator.

14 Programming rules

Every programming system has an underlying risk of the programming being done in an unstructured manner, and hence, the outcome that the program is very difficult or, in fact, impossible to comprehend for you as a programmer as well as for the customer or any other person working with the system.

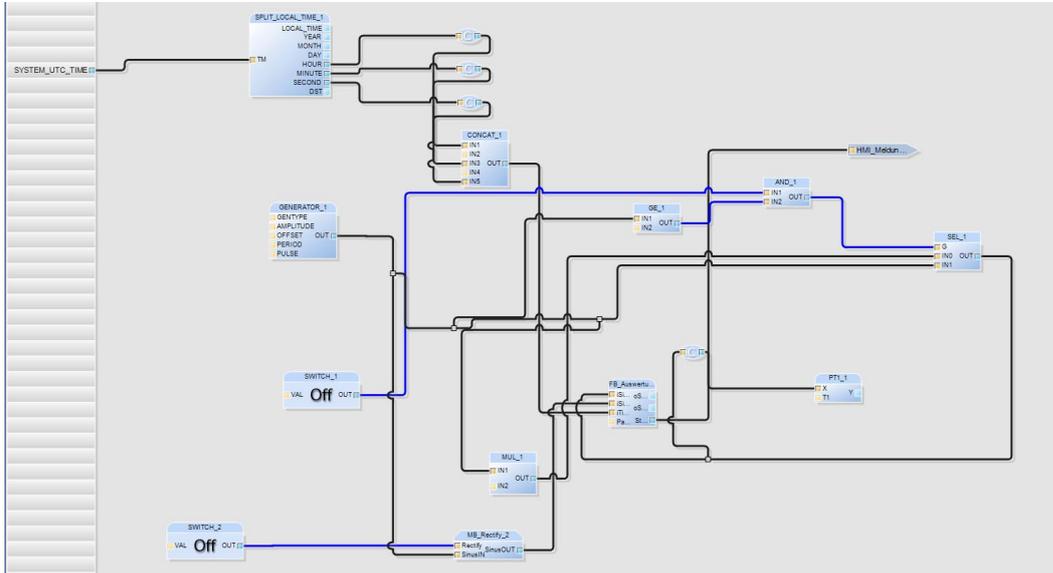


figure 133: Example of unstructured programming

The example in the figure given above is restructured for the recommended solution.

14.1 Approach for the solution

Two tasks having the following structure:

- Task 1 Data generation
- Task 2 Data processing

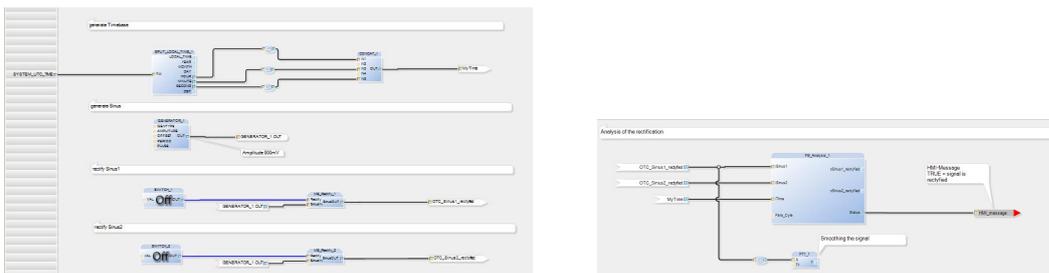


figure 134: Example of structured programming

You do not have to comply with the following guidelines, but however, they simplify working with ibaLogic.

- Distribute the functions across multiple tasks / programs having illustrative names.

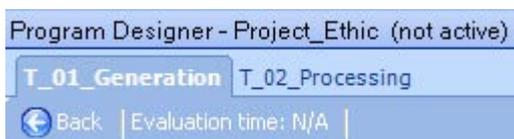


figure 135: Division of tasks / programs

- ❑ Create one task each for the hardware inputs and the hardware outputs. Assign the priorities in such a manner that the input task is the first and the output task the last to be processed.
- ❑ Use intra-page connectors within a task, if too many intersections of the lines make the layout cluttered
- ❑ Tag the sub-functions using comment fields.

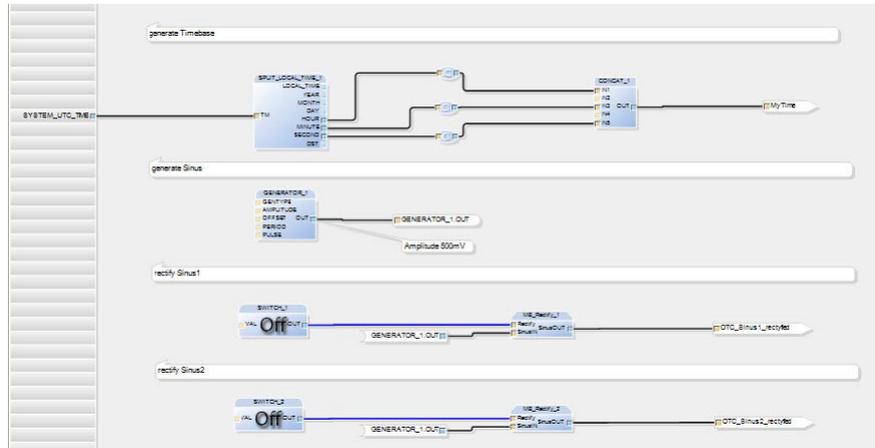


figure 136: Comments

- ❑ Merge reusable program components into macro blocks and create meaningful description and comments for them.
- ❑ Combine complex connections pertaining to a function into a macro in order to improve overall clarity.
- ❑ Use comments and descriptions even within an FB. We recommend a header with a change index, titles and meaningful indentations of the program lines.

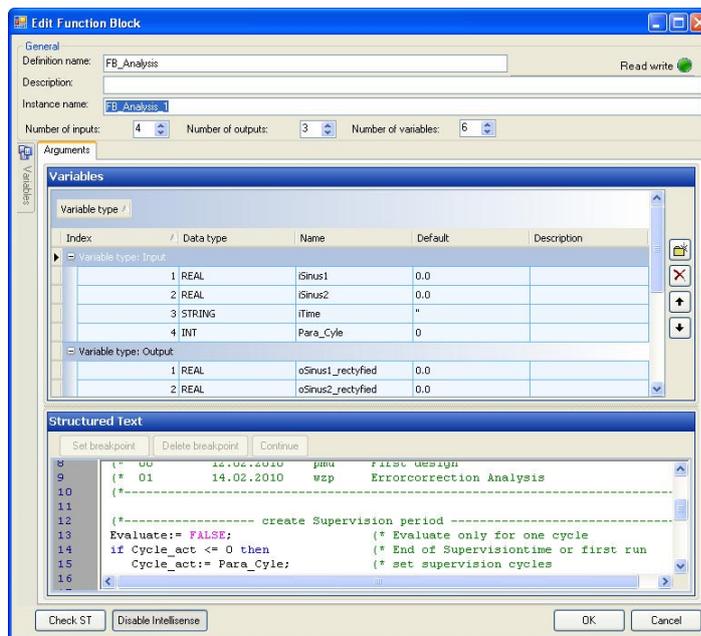


figure 137: Program code comments

- ❑ Arrange the blocks within a task in such a manner that they match the evaluation sequence (from the top left to the bottom right).

- ❑ Designate off-task connectors with a prefix, e.g. "OTC_" or, if the OTC is used for an HMI system, with "OPC_".
- ❑ Designate the intra-page connectors with the prefix "IPC_". If an "OTC_test" is present as an input, it can continue to be used internally as "IPC_test" and there is no repetition of names.
- ❑ Configure short prefixes for the names of blocks, macros and their connectors, e.g. "FB_", "MB_".
(Setting under "Tools – Options – Editors – Function Blocks").
- ❑ Assign names to user-defined data types such that they give an indication of the logical meaning (e. g. ST_ROLLING) or the contents (e. g. AR_64REAL).
- ❑ If necessary, move the lines in such a manner that you can trace them clearly. Avoid any overlapping.
- ❑ Utilize the option of enlarging the blocks as desired. In doing so, connector names become more legible or the lines are easier to trace.

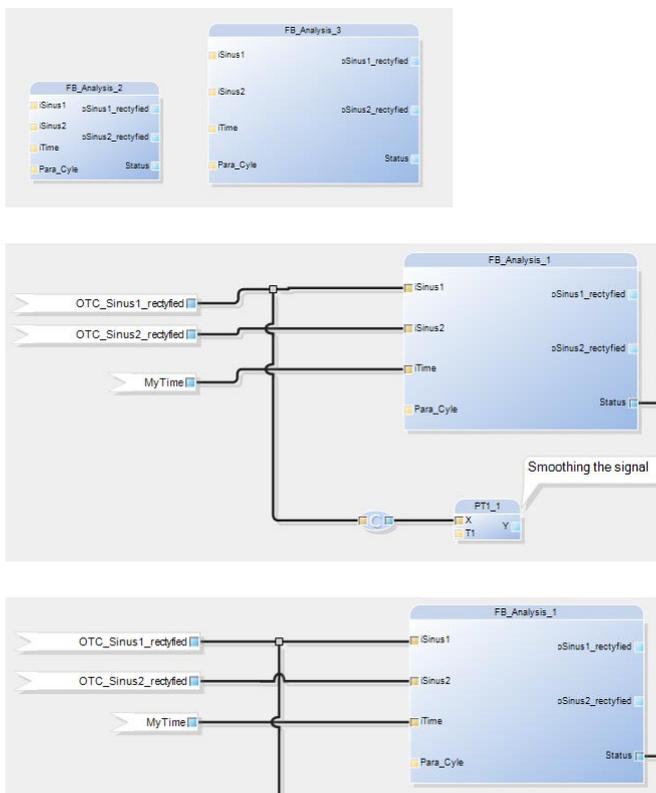


figure 138: Examples of enlarged display

- ❑ In general, trap possible sources of error while programming, such as:
 - Division by 0
 - Access beyond the array limits
 - Possible endless loops

15 Uninstall ibaLogic

⚠ DANGER

During uninstalling, various messages can occur:

- Query whether the user backups are to be deleted, too.
 - Query whether SQL Express is to be uninstalled, too (only occurs if the ibaLogic database existed exclusively).
-

⚠ DANGER**Danger by enabling or disabling functions!**

Possibility of human injuries and damage to machinery by enabling or disabling functions and other services (PMAC, OPC ...), which have a direct impact on the response of the system.

Secure the system while working on it!
Follow the safety regulations applicable!

**Important Note**

Only those users having administrator privileges can uninstall ibaLogic software. Please ask your system administrator.

Prerequisite

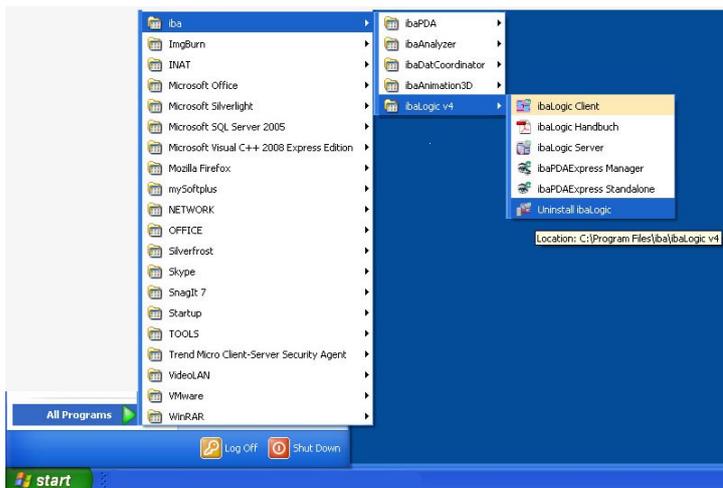
- All ibaLogic programs are closed.
-

**Note****Messages during uninstalling**

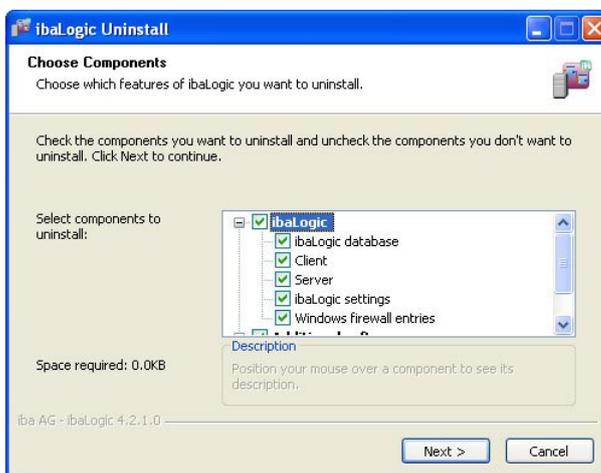
SQL Express Instance is removed by confirming the prompt with <Yes>. The database created during installation is deleted (*.ldf, *.mdf).

Procedure

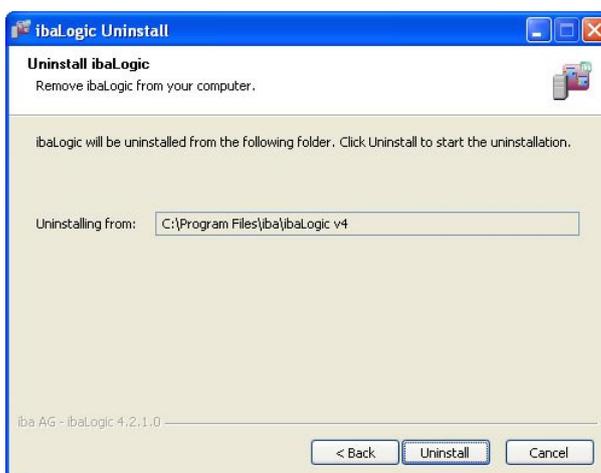
1. Select "Start – iba – ibaLogic v4 – uninstall ibaLogic".



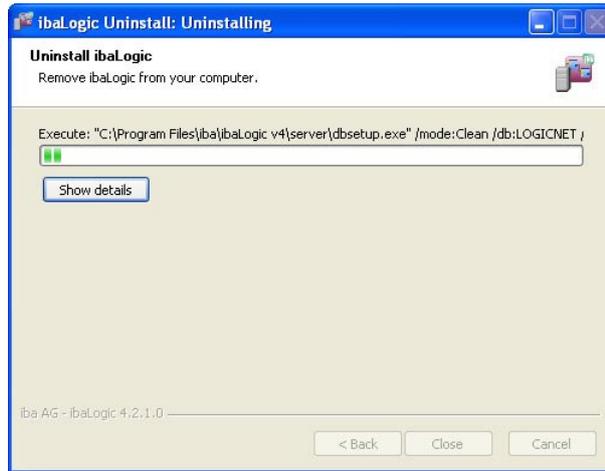
2. Choose the components that you wish to uninstall.



3. Start the uninstalling by pressing the <Uninstall> button.



4. Close the dialog box by pressing the <Close> button. Confirm any confirmation prompts if required.



Important Note

If there are any database backup copies in the installation folder, you are asked during uninstall whether these should also be deleted. If you confirm with <No>, the backup copies remain.



16 Practice Examples

The aim of this section is to accompany the "Beginner" with the first steps in using ibaLogic.

It is intended that a small sample program generates the "Aha Effect" and, above all, demonstrates what iba understands of ergonomic CFC implementation, and what meaning is given in the process to online update of static and dynamic variables.

Since this is merely an introductory example, it does not illustrate or deal with all functions of ibaLogic.

16.1 First Steps - Sample Project

The task is to create a program with which a sinusoidal signal is generated. A smoothly adjustable offset should be added to this sinusoidal signal depending on the status of a switch. You can create the example completely with the help of standard function blocks. Nonetheless, in order to be able to explain the highly flexible programming opportunities and features of the integrated programming language, "Structured Text" (ST), the example should also be programmed using this alternative.

The input signals and the result need to be displayed as trend graphs.

The sample program is made "live" so that the connectors used are updated continuously. The change of color to pink indicates that everything is now "serious": The plant is virtually live!

If outputs are already connected (Actuators, motors, etc.), these would respond immediately to the program modifications. The customary procedure - programming, compiling, linking and loading, as well as starting - runs automatically in the background. Moreover, for the sake of simplification, the preset values for the project and program names are accepted.

16.1.1 Sample Exercise Part 1

16.1.1.1 Task Description

The periodically changing value of a generator (Sinusoidal signal) needs to be added to the value of a slide controller depending on the status of a switch:

Switch position 1: Generator + Generator

Switch position 2: Generator + slide controller

All variables and, of course, even the result needs to be displayed in a graphics format as a trend curve.

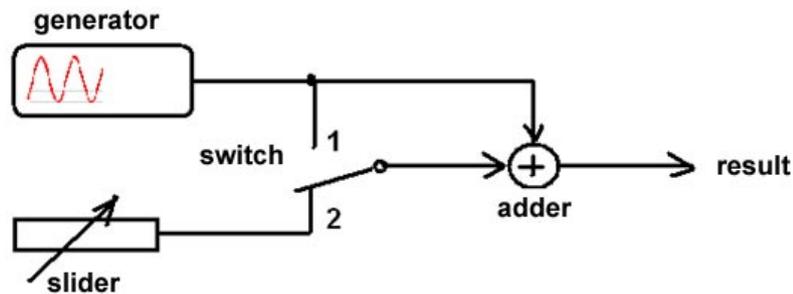


figure 139: Circuit diagram of the sample exercise

In the first part of the exercise, the example should be implemented with the help of function blocks (FB) that are available as standard blocks in ibaLogic.

Since laboratory equipment such as a function generator, slide controller and keys do not have to be connected to the inputs of ibaLogic, such effective functions have been compiled as **Specials** and can be placed like other blocks. You can perform active operations on the **Switch** and the **Slider** to test the circuit.

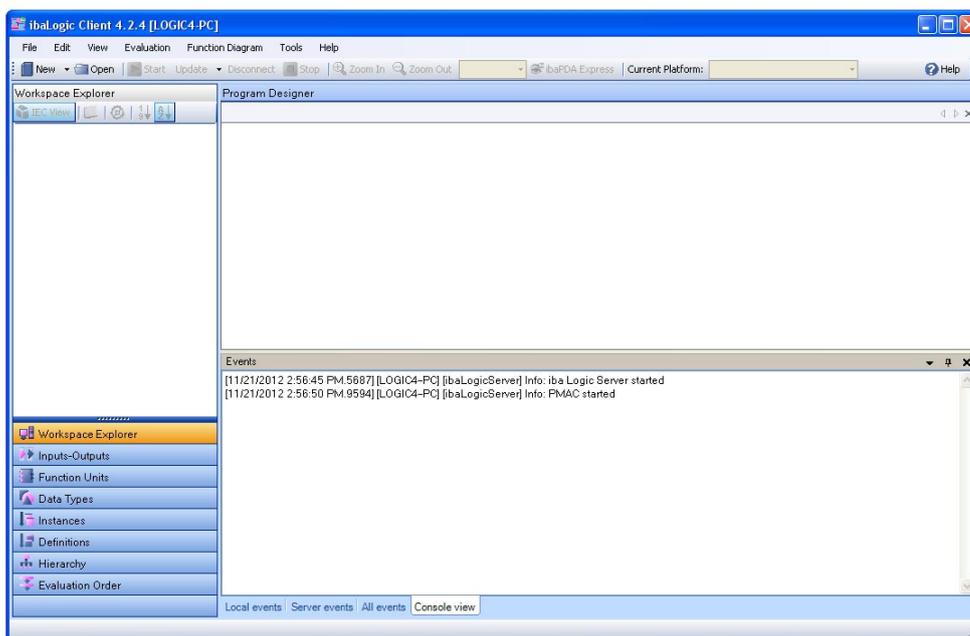
16.1.1.2 Start ibaLogic Server and ibaLogic Client

Procedure

1. Double click on the "ibaLogic Server" icon on the desktop.
The ibaLogic Server dialog box opens after the initialization phase. By default, the server is started automatically when the dialog box opens.



2. Double click on the "ibaLogic Client" icon on the desktop.
The ibaLogic Client dialog box opens after the initialization phase.



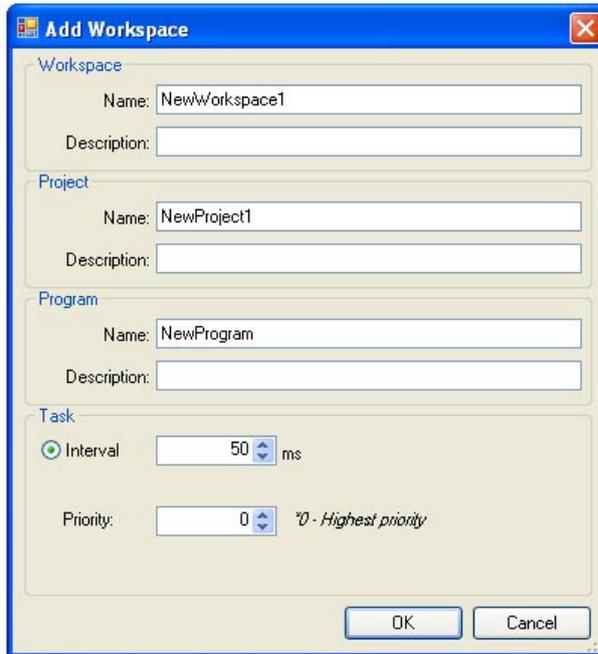
Remarks

The event window below the program window documents the program actions and collisions, if any. If error messages pop up during the startup of the server or the client, please refer to this documentation for assistance.

16.1.1.3 Create a New Project

Procedure

1. Press the <New> button in the toolbar.
The "Add Workspace" dialog box is displayed.



2. Confirm the entries with <OK>.

When you do not change the presets, your project is called "NewProject1" having just one program "NewProgram". The preset interval time of 50 ms is adequate for the example.

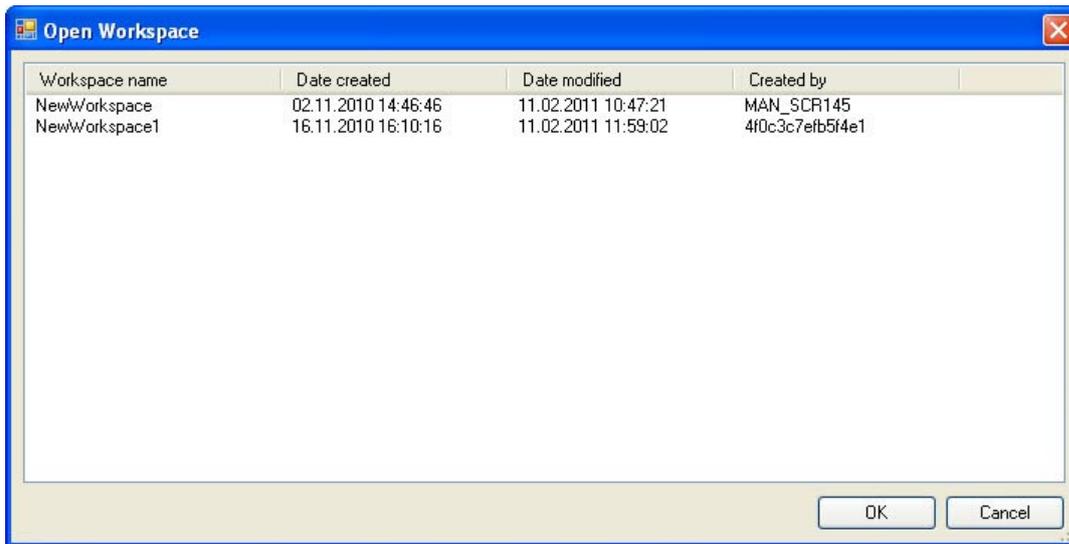
Remark

If, after beginning your exercise with ibaLogic, you have had to interrupt the "Session" or you have simply closed all programs (first the client and then the server), you do not have to begin with <New>.

Your changes are saved automatically.

When you continue in such a case, press the "Open" button, open the workspace you created and continue working at the position where you have stopped.

If you have several workspaces, the search function can be limited by the "Modification Date" field.



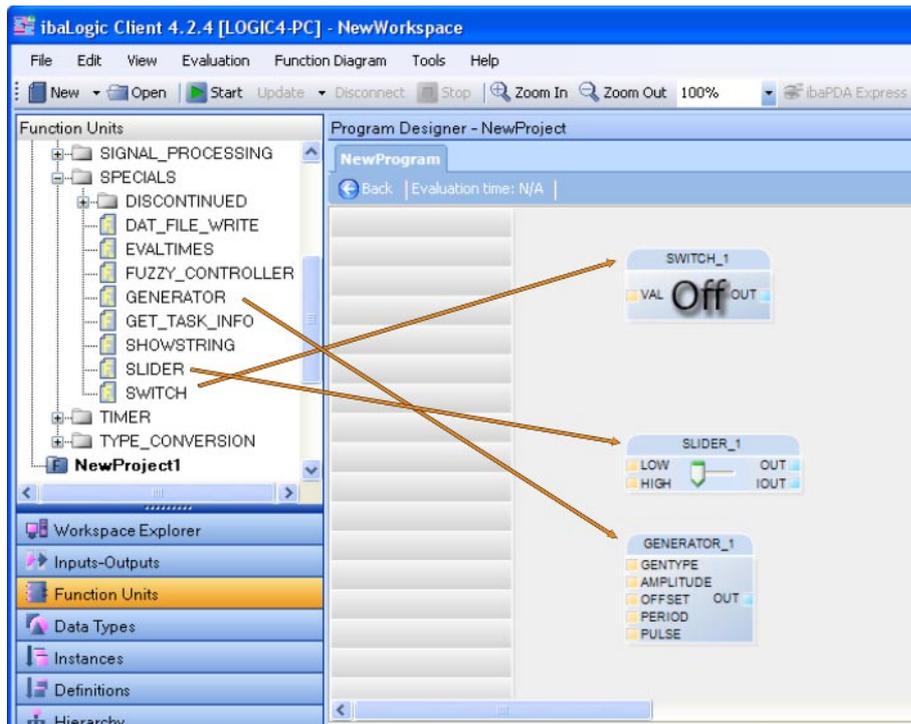
16.1.1.4 Placing the Test Tools

One function block each is required for the task description:

- "Generator"
- "Switch"
- "Slider"

Procedure

1. Click on the <Function Units> button. A folder tree opens in the navigation area.
2. Open the "Specials" folder.
3. Drag one "Generator", one "Slider" and one "Switch" to the design area of the program designer keeping the left mouse button pressed.
If, in the process, you come too close to a block that is screened, a superimposed shadow indicates its "Sphere of intimacy" in which no other block should be placed.



4. Arrange the blocks in the proper sequence.

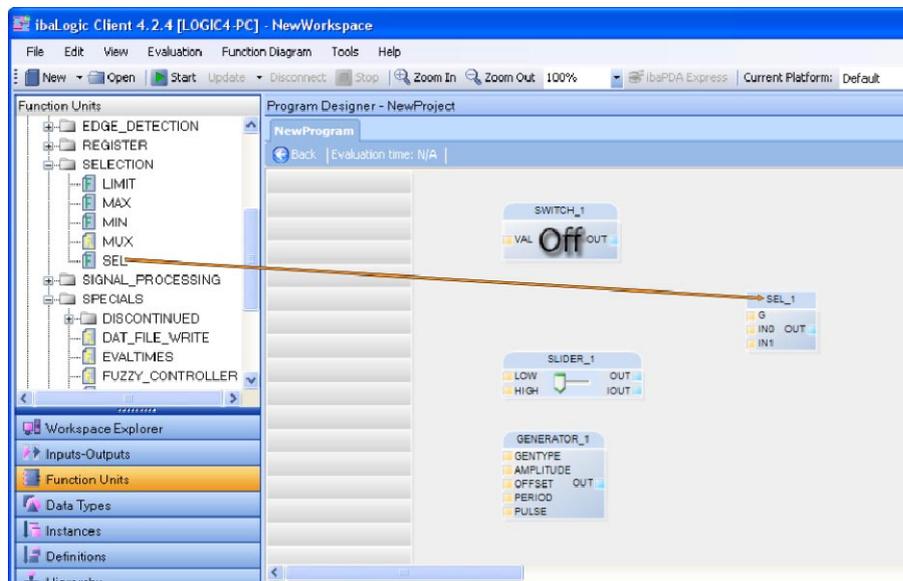
16.1.1.5 Placing the evaluation blocks

One function block each is required for the task description:

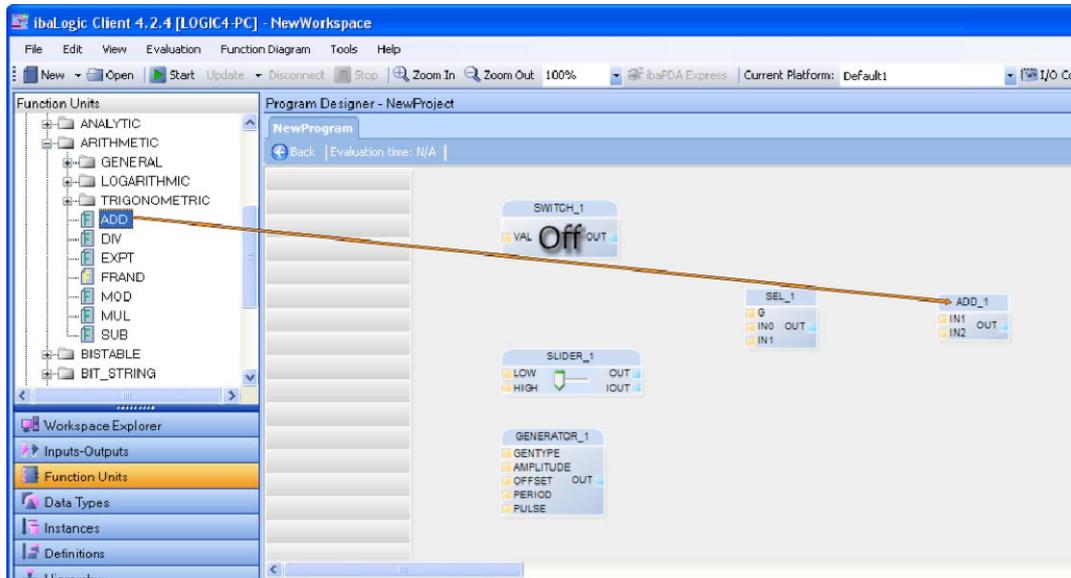
- "Selector"
- "Adder"

Procedure

1. Open the "Selection" folder in the directory tree of the navigator.
2. Drag the selector (SEL), keeping the left mouse button pressed, to the design area of the program designer.



3. Open the "Arithmetic" folder in the directory tree of the navigator.
4. Drag the adder (ADD), keeping the left mouse button pressed, to the design area of the program designer.



Remarks

The SELECTOR (SEL) requires a binary signal as the "Decision-maker" (Selector). The values IN0 and IN1 under consideration for the selection are format-free and they adapt themselves automatically to the data type with which they are connected.

As already described with the selector, the values to be added are format-free to begin with. It is only the connection that is first "joined" to one of the inputs that determines the format.

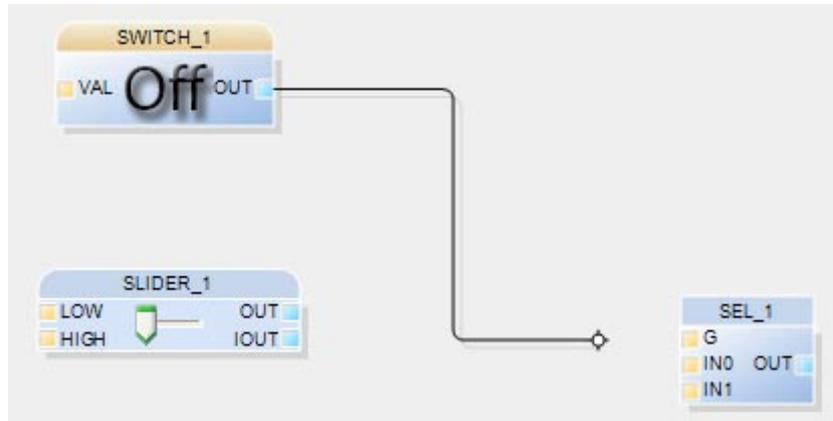
16.1.1.6 Connecting the selector block with the test tools

The following need to be connected:

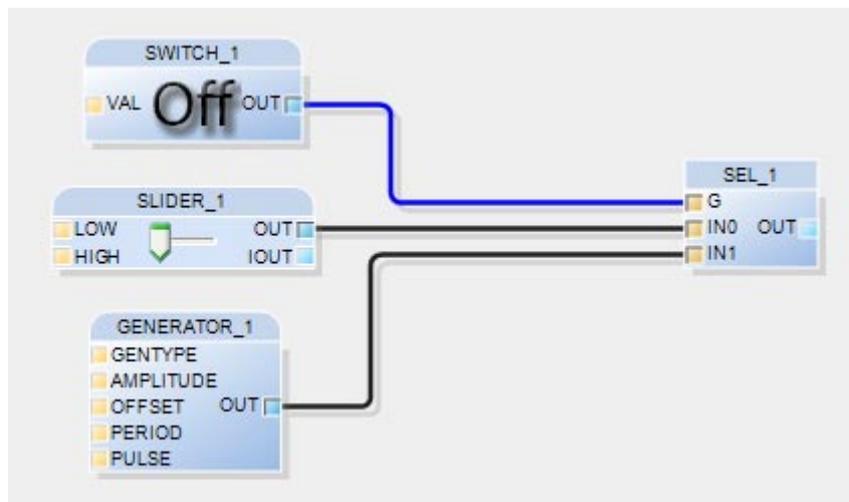
- The switch output (SWITCH) OUT with the decision-making input G of the elector (SEL).
- The slider output (SLIDER) OUT with the input IN0, of the selector (SEL).
- The generator output OUT with the second input IN1 of the selector (SEL).

Procedure

1. Place the mouse pointer on the connector (OUT) of the switch and drag the mouse pointer, keeping the left mouse button pressed, to the connector (G) of the selector.



2. Complete the remaining connections accordingly.



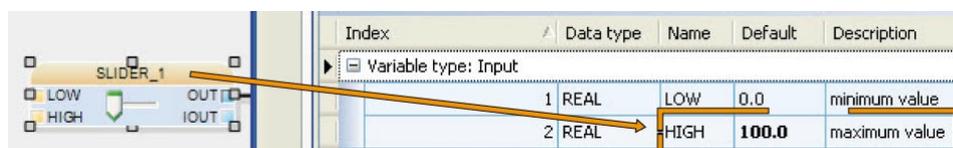
16.1.1.7 Configuring the slider and generator

The following need to be configured:

- "Slider"
- "Generator"

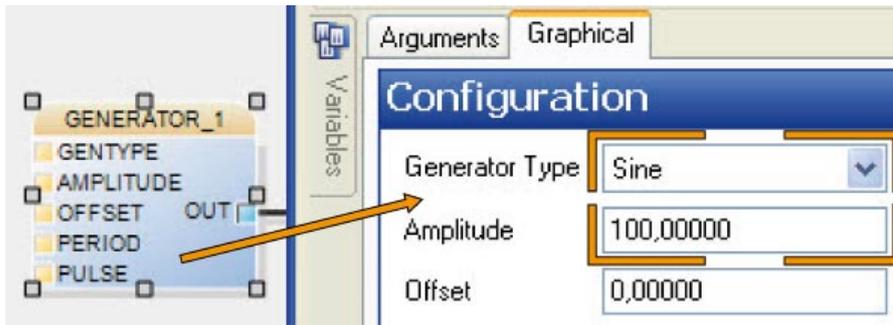
Procedure

1. Double click on the slider. The configuration menu pops up.
2. Set the "maximum value" of the working range to 100.0. The slider works with a value between 0 and 100 depending on the slider position.



3. Open the configuration menu of the generator.

4. Set the generator type to sinusoidal.
5. Enter an amplitude value of 100.



With this setting, the generator produces a sinusoidal waveform having a value between +100.0 and -100.0.

Leave the period of the oscillations, as preset, at 10 sec.

16.1.1.8 Switch the partial connections online

The online compiler is also activated by starting the evaluation.

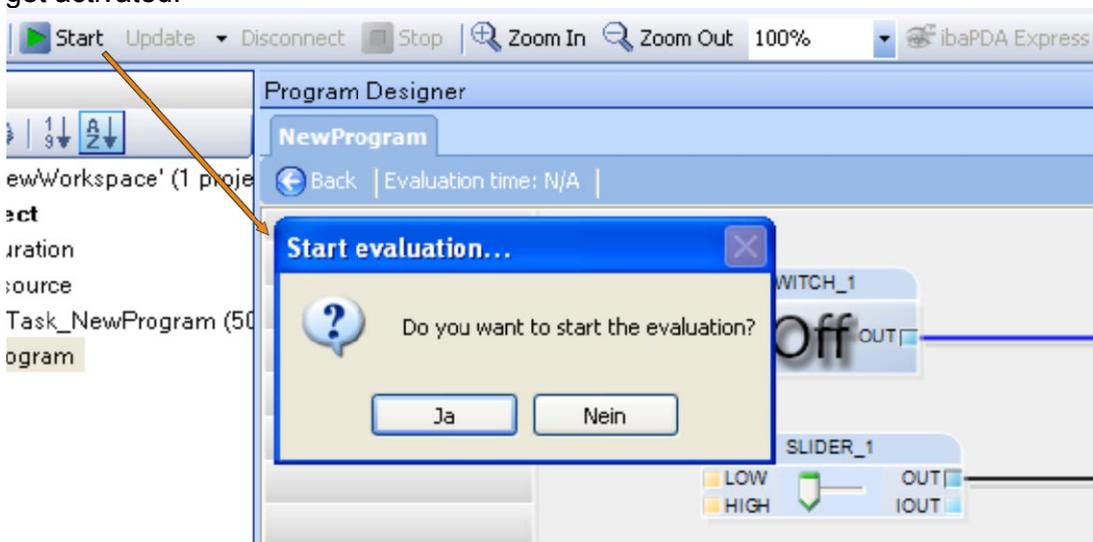
- The background color of the program designer changes to pink.
- All binary connecting lines are colored depending on their status.
- The block connectors are updated continuously and displayed.
- All connections are now "live".

Figuratively speaking, this is comparable with applying a voltage to the test circuit.

In a real-life situation, machines are controlled by ibaLogic via the outputs connected. In such a case, the impact of "Connecting live" in the case of a program bug would be felt drastically.

Procedure

- Click on the <Start> button in the toolbar to start the evaluation of the sample exercise. Confirm the prompt with <Yes>. The online properties mentioned above get activated.



16.1.1.9 Testing the switch and selector

Since the selector is already connected and, moreover, the circuit is switched online, its function can be tested immediately independent of the switch status of the input (G). The principle of operation of the selectors is described in this manual under the standard blocks. You can now implement these explanations in practice.

Procedure

- ➡ Change the status of the switch by clicking on the SWITCH with the left mouse button.

As you can see, in the switched off status (Off), the slider output (Value = 43.5) is connected to the SEL output (OUT).

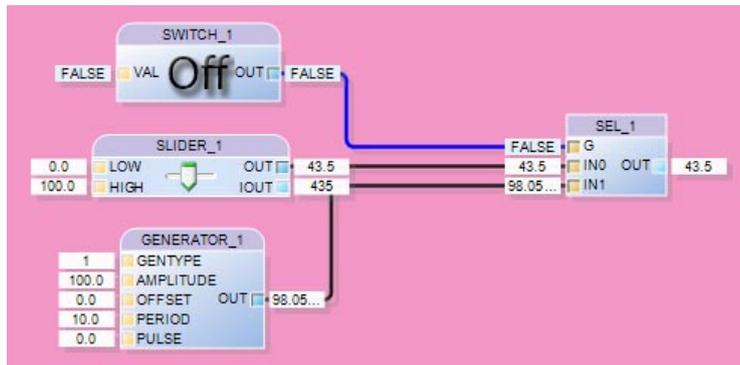


figure 140: Slider output (Value = 43.5) connected to the SEL output (OUT)

In the switched on state (ON), the periodically changing generator output is connected to the SEL output (OUT).

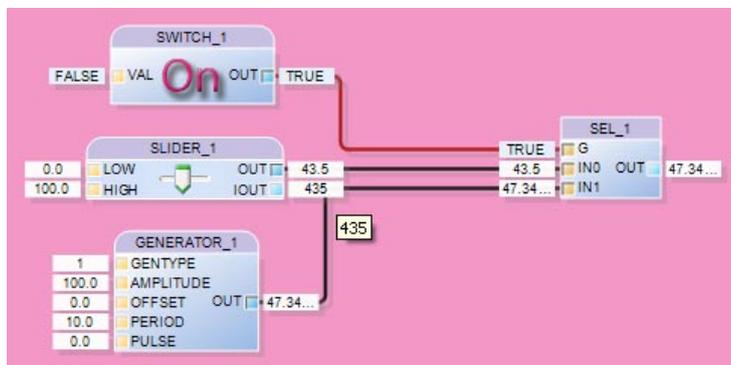


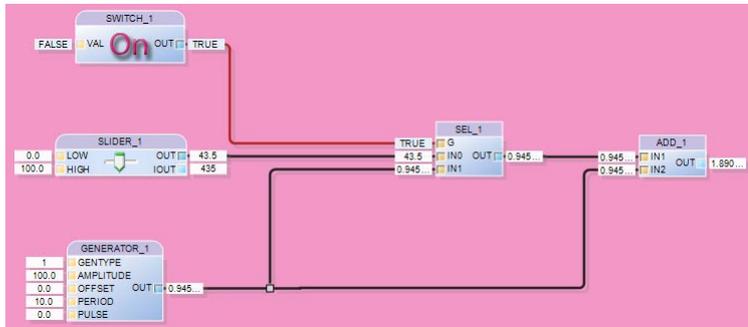
figure 141: Generator output connected to the SEL output (OUT)

16.1.1.10 Connecting the adder

Connect the adder in order to complete the circuit as given in the sample exercise.

Procedure

1. Place the mouse pointer on the selector output (OUT) and drag the mouse pointer, keeping the left mouse button pressed, to the input (IN1) of the adder.



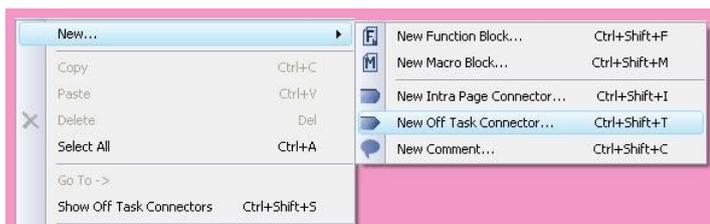
2. Place the mouse pointer on the input (IN2) of the adder, and drag the mouse pointer, keeping the left mouse button pressed, to the generator output (OUT). The connection point above the mouse pointer moved locks in position automatically and forms a branch.

16.1.1.11 Create an OTC to illustrate the result

In our sample exercise, an off-task connector is placed and also connected to illustrate the result of this simple example. The OTC is called "Result".

Procedure

1. Click with the right mouse button on the design area of the program designer.
2. Select "New... – New Offtask Connector...".

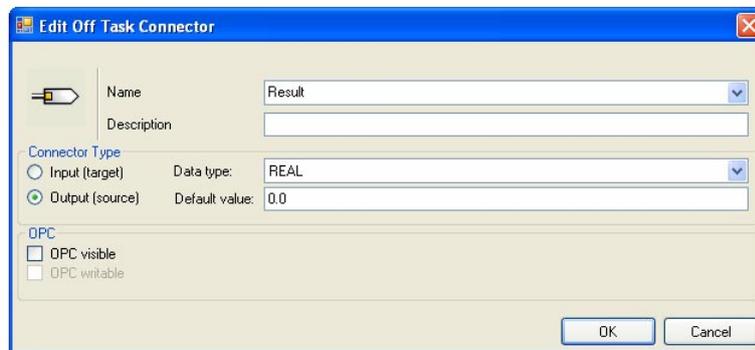


The window "Off-Task connector edit dialog" is displayed.

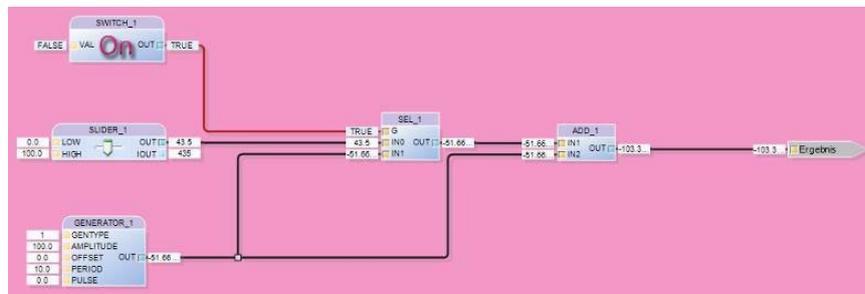
- Assign the name "Result" in the input field. All other preset values remain as they are. Configure the data type as "REAL" and set a default value of 0.0.

The preset value as output is correct, since a value is fed to the OTC.

The objective would be the OTC, if it were to receive a value transferred from another program via the OTC.



- Connect the OTC "Result" with the adder.



Remarks

An OTC is a key element in graphics programming.

The OTC facilitates the clarity of a project by virtue of the fact that it is distributed over several programs that communicate with one another via OTCs.

In contrast, the inter-page connector is used within a program.

In this manual, you will also find "Programming rules". This section gives ideas about how you can make use of OTCs and IPCs in order to avoid tapeworm programming (entangled connections).

A highly significant communication element is the OTC for connecting with a supervisory HMI (Human-Machine Interface) system.

16.1.1.12 Analysis of the circuit

You can control the result with the help of the dynamic display of the connectors. However, even in this simple program running with a 50 ms cycle time, checking the numerical results is extremely difficult.

Please pay attention to the "Sample Exercise Part 2, Page 260" in the following.

16.1.2 Sample Exercise Part 2

This sample exercise demonstrates the comfort of a dynamic online trend curve display to evaluate the result.

16.1.2.1 Program analysis using the ibaPDA Express

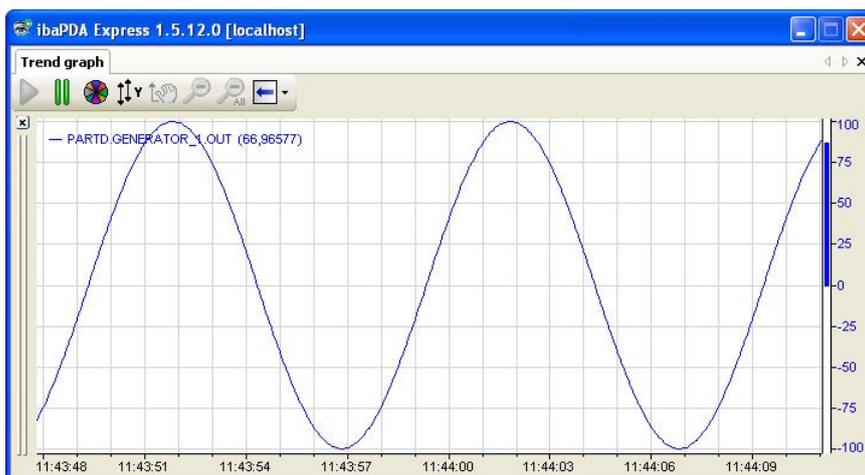
An ibaPDA display is integrated into ibaLogic for online display of trend curves.

Procedure

- Click on the integrated ibaPDA Express in the toolbar.
This program is integrated in every version of ibaLogic at no extra cost.

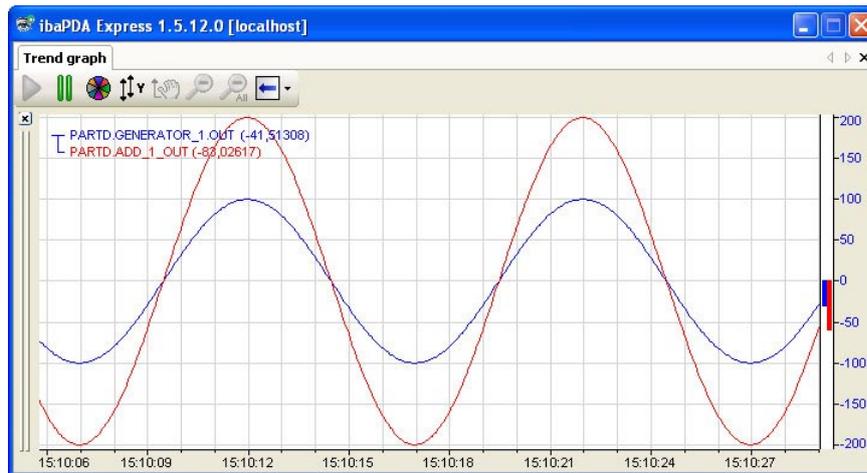


- Move the mouse pointer to the OUT connector of the generator and drag it keeping the <ALT> button pressed and drop it in the ibaPDA Express window.
- Track the intermediate value selected (Generator OUT) as a trend curve display.
- Adjust the Y-axis using the <Auto scale All> button.



- Drag the intermediate values and results of interest to you into the ibaPDA Express window.
 - If these are variables having the same scale, the vales get dragged to the connector text of the signal already being displayed.
 - If the scales are different, drag and drop the connectors into the trend curve window and a new scale is formed.
 - In order to open a new trend curve window, pull the connector to an area outside the current trend curve window.

- Carry out a final check on the sample exercise by changing the switch position to on / off. Observe the changes in the trend curve display.



16.1.3 Sample Exercise Part 3

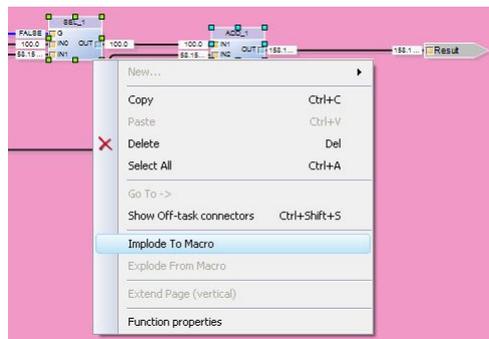
Improving the program clarity and readability

You can combine function blocks to macros in order to improve the clarity and readability of a draft circuit.

16.1.3.1 Procedure

This technique is demonstrated with this small sample exercise.

- Mark the FBs under consideration and their connecting lines with the <Ctrl> button pressed at the same time. In this case, mark the selector and adder, and their connecting line.



- Click the right mouse button with the <Ctrl> key pressed, which displays the macro menu.
- Select the "Implode To Macro" in the macro menu.
- Confirm the intermediate screen (Macro, inputs, outputs).

16.1.3.2 Remark

A macro is created. The macro created with a preset name (IMPL_MB_1) performs the same functions as the individual blocks selected previously. However, it can be designated appropriately for meaningful and proper documentation.

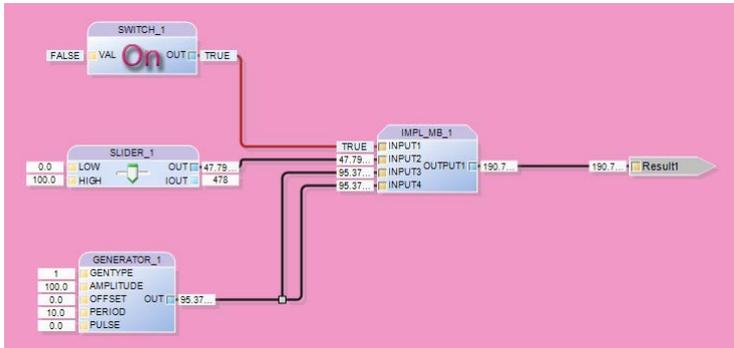


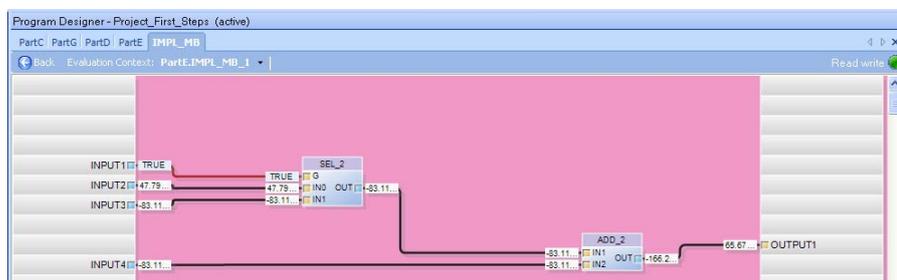
figure 142: Macro creation



Tip

You can use the lasso method to combine a larger number of FBs into one macro. Keep the left mouse button pressed and mark a rectangle across the desired objects.

- Double click on the macro generated to display its contents. Some of the lines or the FBs are placed in a somewhat haphazard manner. You can also edit in the macro, e. g. add connecting lines that you have overlooked or only "clean it up" graphically.



- Select <Back> to quit the macro zoom display.

16.1.4 Sample Exercise Part 4

Creating function blocks using ST

The function blocks provided as standard blocks are highly comprehensive in accordance with the standard. Based on the experience of their own applications and, above all, based on the suggestions of ibaLogic users, some other useful and special FBs have been developed and included as standard blocks.

In order to demonstrate to the users the options and methods of creating special blocks on their own, one FB has been programmed at the end of this sample exercise. As a functional test, this new FB runs in parallel to the macro created earlier. The results must be congruent with one another.

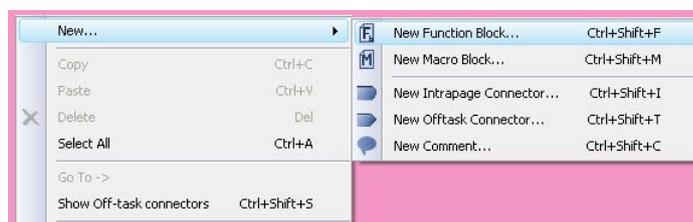
The IEC 61131-3 standard has given special importance to the "Structured Text" programming language. With the help of this higher-level language, you can program very clear and easily readable FBs along the lines of the Pascal programming language. The ST blocks can also be ported to third-party systems.

If you come from the classical PLC world, you are already conversant with the use of standard FBs.

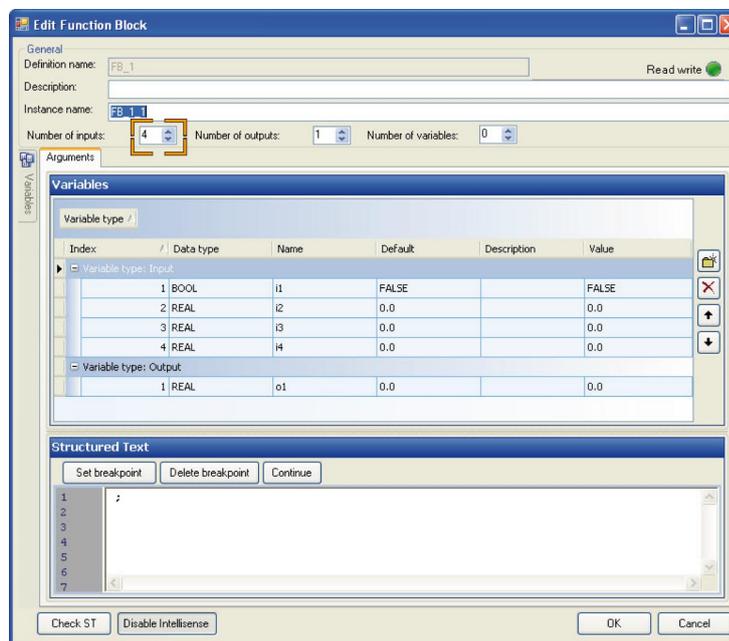
On the contrary, if you are entering this kind of the controller world as an Assembler or high-level language programmer, you would probably find it easier to deal with Structured Text programming.

16.1.4.1 Procedure

- Select "New... – New Function Block..." in the context menu. The "Create Function Block" dialog box is displayed.



- Enter "FB_1x" as definition name.
- Set the "Number of Inputs" to 4. Since the macro from the sample exercise part 3 needs to be cloned, the FB needs as many inputs- and outputs (4- inputs and one-output).



- Configure the data types for the inputs and the output.

For the inputs

- 1x BOOL
- 3x REAL

For the output

- ❑ 1x REAL

Using the button "", you create a new variable.

- ➔ Enter a semicolon in the "Structured Text" field.

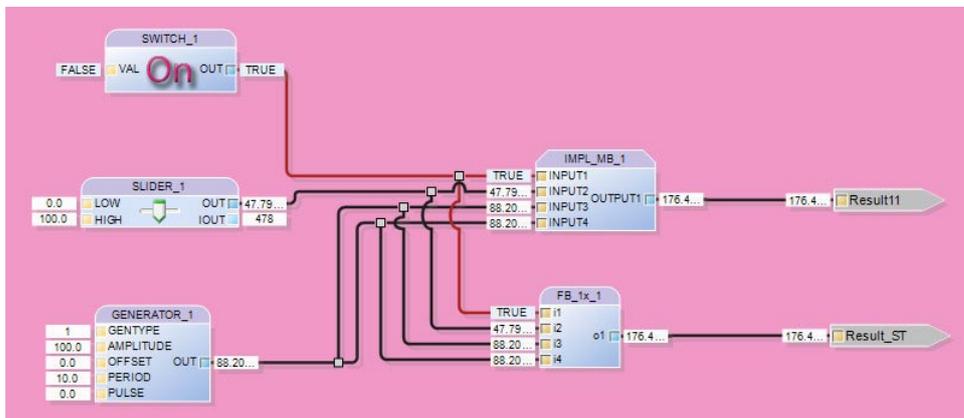
16.1.4.2 Remark

In the first approach, merely an "empty" function block is created.

The ST field must contain at least one semicolon on formal grounds.

Since the inputs and outputs have already been defined, the new FB can be added in the layout and can also be connected in parallel to the existing macro.

- ➔ Place the new FB appropriately in the layout.
- ➔ Connect the new FB in parallel to the macro. In order to create the second result OTC, copy the existing one with <Ctrl+C>. Add it by pressing <Ctrl+V>. Assign a new name to the OTC, e. g. "Result_ST".

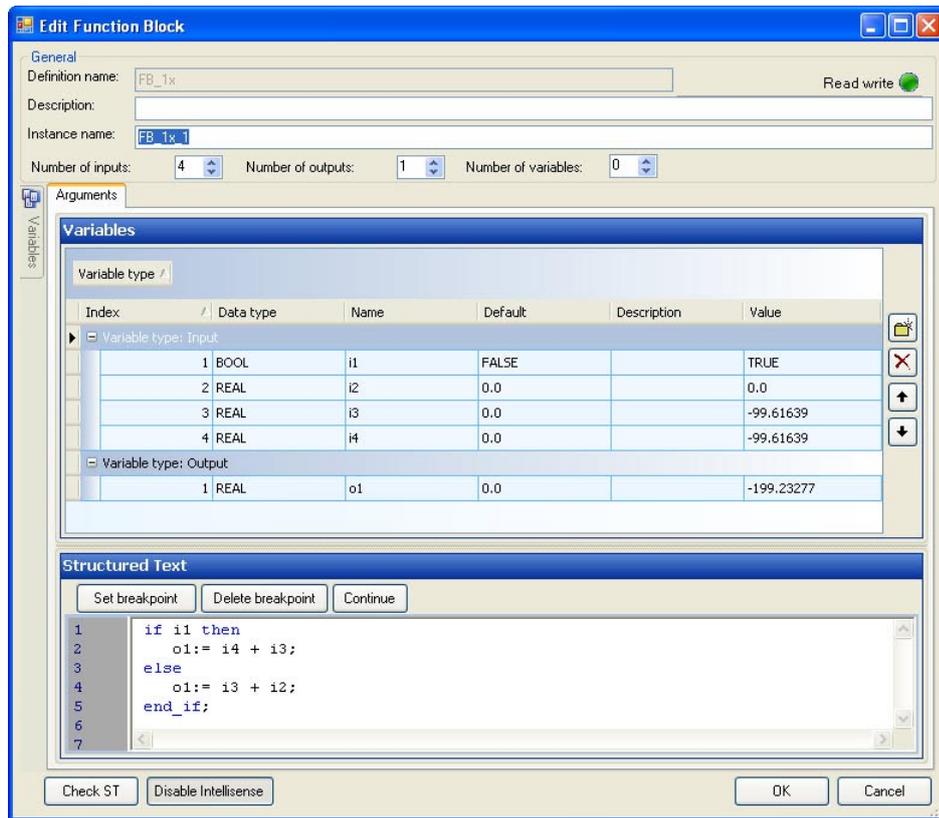


- ➔ You can see that the FB is not yet responding to its inputs. This still needs to be programmed for its task.
- ➔ Double click on the slider. The "Edit Function Block" window is displayed. You can now carry out the programming using the customary "if", "then", "else" and "end_if" statements available in almost all high-level languages.
- ➔ Enter the following source code (as illustrated in the below screenshot) in the "Structured Text" field.

```

1 if i1 then
2     o1 := i4 + i3;
3 else
4     o1 := i3 + i2;
5 end_if;

```



16.1.4.3 Result

The example begins with the inquiry of the switch status (if i1 = "TRUE" or shorter, if i1) and adds accordingly, the inputs i4 + i3 or i3 + i2.

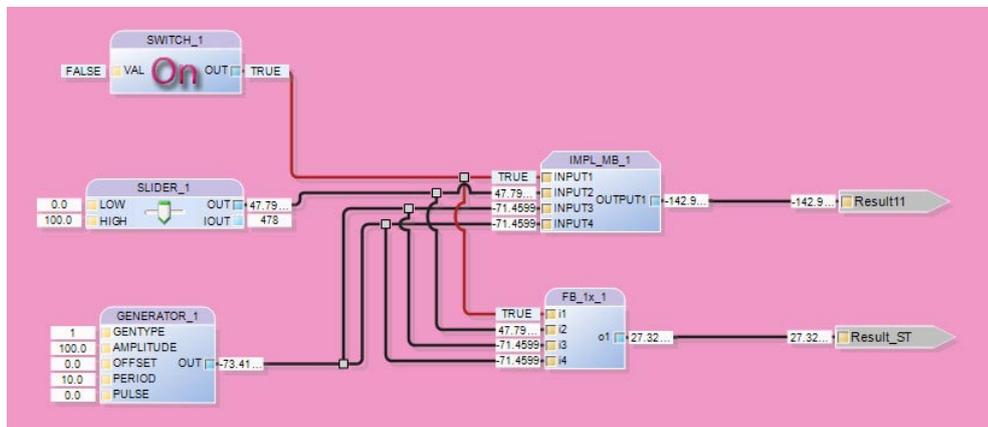


figure 143: Sample circuit with addition inquiry

16.1.4.4 Remarks

Please also note that all variables in the block are updated online!

- Drag the result of the ST block with the same scale into the ibaPDA Express display.

The second red curve is congruent with the blue curve (Result from the macro). Only the last color (blue) of curve 2 is visible. You can see from the bar displayed on the right that there are two individual values.

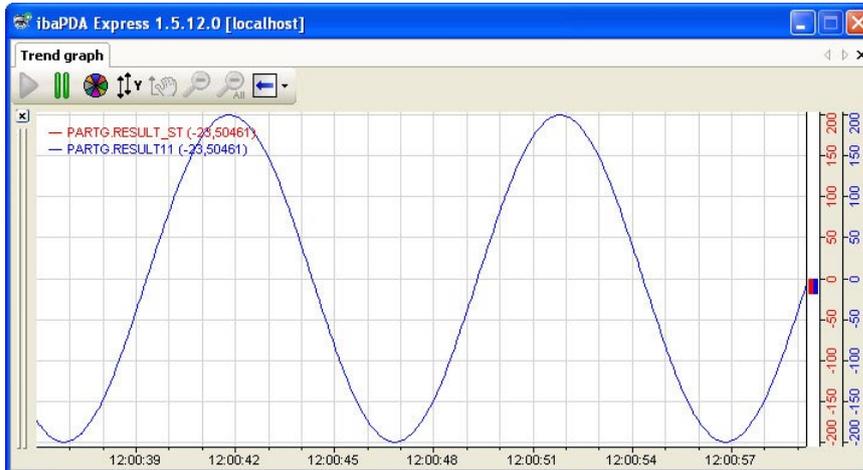


figure 144: Congruency of the results

16.2 DAT_FILE_WRITE Sample Project

16.2.1 DAT_FILE_WRITE in "Unbuffered" Mode

In the "Unbuffered" mode (Explanation see "Buffered Mode, Page 192") one data sample is stored in each storage cycle.

Use this mode when the sampling cycle of the data to be saved matches that of the ibaLogic task cycle. Or when you wish to save data that is generated within ibaLogic itself.

Task: Store 8 analog values and 8 digital values from ibaLogic

Preset parameters

- Platform: Win XP
- Task interval: 20 ms

16.2.1.1 Step 1: Configure the DFW block

➤ General Configuration

Asynchr. access:	Disabled
Storage cycle:	10 (not relevant)
Start time offset:	0
Save values:	Enabled
Write to file:	Disabled (is controlled externally)
Post-processing:	Disabled
Sign the file:	Enabled
Technostring:	Empty
File information:	Empty
Folder:	Choose a directory on a local drive using the browser button, e. g. "d:\dat\ibaLogic\"
File name:	Accept the default value specified.
Sampling time:	Specify the task interval time: „0,02" s

➤ Signal configuration

Name:	"module_01"
Mode:	„Unbuffered"
Values:	1 (not relevant)
Digital values:	8
Data type:	REAL
Analog values:	8



- ➔ Click with the mouse in the signal definition area. In doing so, the module is created with 8 digital and 8 real values. The names are preset with "Digital_xx" and "Analog_xx".

You can rename the standard signal names, e. g. to "Sine".



16.2.1.2 Step 2: Connection of the DFW

- ➔ End the module editor.
- ➔ Join the connection "STORE_FILE" with the output of a "SWITCH" block.
- ➔ Join the first measured signal, e. g. the output of the sinusoidal generator (Data type REAL) with the input connector "DATA".
 - A selection menu opens from which you can choose the module to which you wish to connect the measured signal.
 - When you move the mouse pointer to the desired module, another selection menu opens from which you can choose the desired signal (e. g. Sine).
 - Another selection menu is opened. Select the menu option "data: REAL".

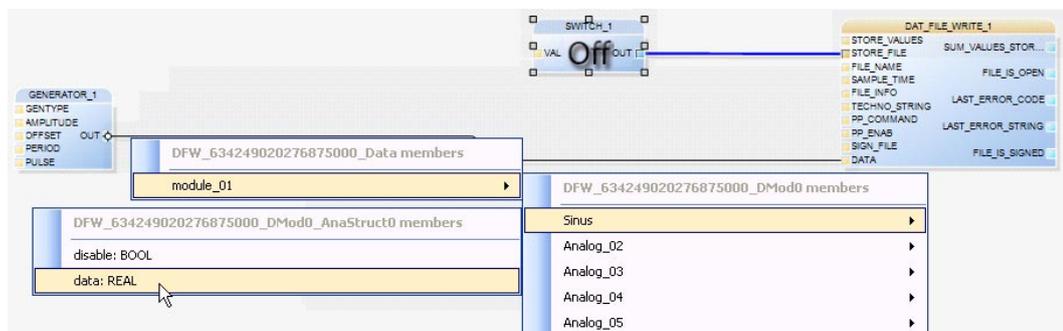


figure 145: Measured signal assignment



Note

Do not let yourself get annoyed by the name of the data structure generated internally. You can ignore it (provided you do not try to reprogram the joiner in the ST, see below).

Result

Based on this, ibaLogic generates the joiner with which the element selected is addressed from the data structure generated internally.

For detailed description of joiners and splitters, please refer to section "Converters, splitters, joiners, Page 161".

The following "joiners" are generated:

- "Module Joiner"
- "Signal Joiner"
- "Signal Property Joiner"

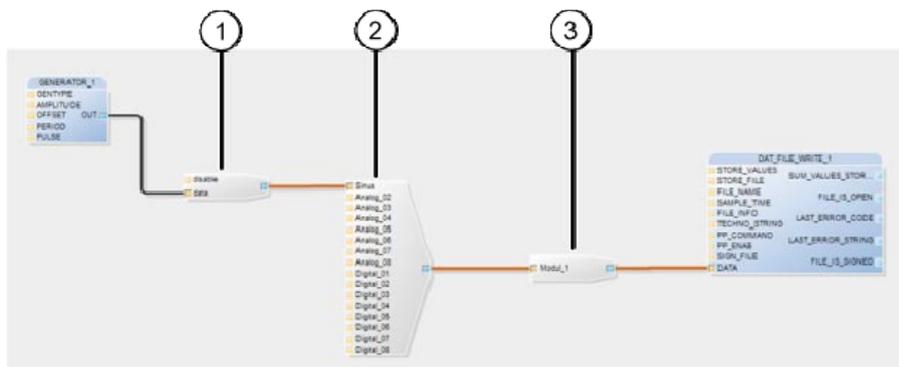


figure 146: Joiner

- | | | | |
|---|------------------------|---|---------------|
| 1 | Signal Property Joiner | 3 | Module Joiner |
| 2 | Signal Joiner | | |

16.2.1.3 Step 3: Create other measure signals

You can connect the other signals directly with the signal joiner.



Note

Do not use the "Disable" inputs of the "Signal Property Joiner" in order to switch off the recording of individual signals for a period of time. This leads to incorrect recording.

Since the individual samples do not have a time stamp, the trend curve is always displayed contiguously. The gap desired is then at the end of the .dat file.

16.2.1.4 Step 4: Starting the recording

- Start the PMAC.
- Set the SWITCH in DAT_FILE_WRITE.STORE_FILE to "On".

Result

You recognize the ongoing recording from the incrementing value of the interface "DAT_FILE_WRITE_1.SUM_VALUES_STORED". Check the result by opening the .dat file generated with "ibaAnalyzer".

16.2.1.5 Alternative: Programming Joiner in ST

In order to keep the layout uncluttered and clear, you can, of course, also program the assignment of the measured signals at the DATA connector in an ST function block.

Example, similar to the configuration given above:

- Go with the mouse to the DATA connector of the DFW module. The tooltip shows you the structure data type generated internally, e. g. "DFW_634094511415156250_Data". Note this down!
- Generate a function block having one or more input variables of type REAL and one output variable of type DATA connector of the DFW module.
 - Activate "Intellisense".
 - Begin with the assignment, and write "o1".
 - As soon as you have entered the point, ibaLogic displays the structure elements (here "module_01"), since "o1" is a structure data type. Select the elements provided using the up/down cursor keys, accept them with "Tab" and place a dot at the end.
 - Since even "module_01" is a structure, ibaLogic displays its structure elements, here, all digital and analog signals. Select "Sine" and place a dot at the end.
 - "Sine", on the other hand, is a structure data type, thus, ibaLogic displays the structure elements "data" and "disable". Choose "data" and continue with the assignment " := i1;".

Result

With this, you have completed the first signal. You can enter other signals in the same manner. The result should then look like this:

```
1 o1.module_01.Sine.data := i1;  
2 o1.module_01.Analog_02.data := i2;
```

You can only connect the inputs with the matching measured signals and the output with the DATA connector of the DFW module.

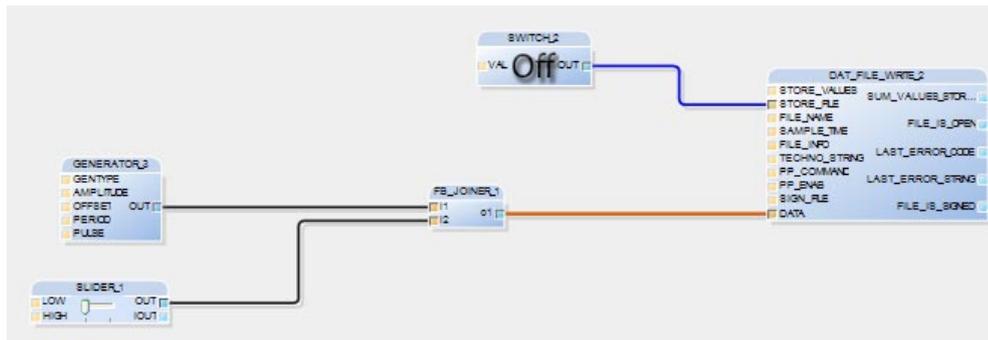


figure 147: Sample circuit "Measured signal assignment"



Note

If you would like to modify the signal configuration of the DFW subsequently, you have to remove the connection at the DATA connector and, after the modification, assign the new data type to the output connector of the FB_JOINER. You may possibly have to modify the assignments in ST.

16.2.2 DAT_FILE_WRITE in "Buffered Mode"

In the "Buffered" mode (Explanation see "Buffered Mode, Page 192") an array of n data samples is stored in each storage cycle. The signals to be stored must be available as arrays. This has the consequence that certain parameters of the DFW module have a slightly changed meaning.

Use this mode when the sampling cycle of the data to be saved is faster than the fastest in the ibaLogic task cycle.

Task: Store 8 analog values from ibaPADU8.

Preset parameters

- Sampling rate of ibaPADU8: 1 ms
- ibaPADU8 to ibaFOB-Link0, mode integer "Buffered"
- Interrupt time base: 1 ms
- Win XP platform
- Task interval: 20 ms

16.2.2.1 Step 1: Configuration of the buffered inputs

➤ General configuration:

The following hardware signals are available for the configuration of data buffering / data transfer (see description in section "Buffered Mode, Page 192"):

- Output signals
"...DataSize", "...Ratio", "...RequestBuffer"
- Input signals
"...CurDataSize", "...FillCount"

➤ Choose the parameters based on the following considerations:

- The array depth in DFW must be greater than or equal to "DataSize".
- Sampling time (in DFW) must be the same as (Array depth * Sample time).
- The "DataSize" must be selected such that the following condition is met:

$$\text{Task interval} \leq \text{Sampletime} * \text{DataSize} / \text{Ratio} / 3$$

(The factor "3" is used to ensure that no samples are lost, even if the task gets suspended.)

- For the example, choose DataSize = 100,
hence, it must be that: "Task interval \leq 1ms * 100 / 1 / 3",
i.e. the task interval must be less than 33 ms. Select 20 ms.

➤ Create a user-defined block, with which you can preset the output signals for the buffered mode, for example

```

1  if (iEnable = TRUE) then
2      if (iEnable <> v1) then
3          oSize := iSize;
4          oRatio := 1;
5          oTime := iTime;
6          oTakeover := TRUE;
7      else
8          oTakeover := FALSE;
9      end_if;
10     oRequest := TRUE;
11 else
12     oRequest := FALSE;
13 end_if;
14
15 v1 := iEnable;
16
17

```

➤ Join the block with the output signals



(The connectors iTime, oTime and oTakeover are not required here.)

16.2.2.2 Step 2: Set the parameters of the DFW module, "General Configuration"

➔ Go offline.

➔ General configuration

Asynchr. access: Disabled

Storage cycle: 10 (not relevant)

Start time offset: 0

Save values: **Enabled**

Write to file: Disabled (is controlled externally)

Post-processing: Disabled

Sign the file: **Enabled**

Technostring: Empty

File information: Empty

Folder: Choose a directory on a local drive using the browser button, e. g. "d:\dat\ibaLogic\"

File name: Accept the default value specified.

Sampling time: Specify the storage interval.
 Storage interval = Array depth * sampling time
 Select an array depth of 200, based on which the storage time works out to 0.2 sec.

➔ Signal configuration

Name: „module_01"

Mode: "Buffered"

Values: 200

Digital values: 8

Data type: Integer

Analog values: 8

➔ Specify the array size as 200 in the "Values" column. This yields a storage time of 200 ms (see above).



➔ Go online.

16.2.2.3 Step 3: Accept the buffered input signals

- Drag a "COLLECT_ARRAY" block from the "Type Conversion" function block folder and drop it in the workspace window.
The block is used to transfer the input array (data type FOBFBUF_INT) into the array for the DFW.
- Create a user-defined (FB_DAAV) having the following properties to generate the "TAKEOVER" signal for the COLLECT_ARRAY:

Variable type: Input			
1	INT	i1	0
Variable type: Output			
1	BOOL	o1	FALSE
Variable type: Variable			
1	INT	v1	0

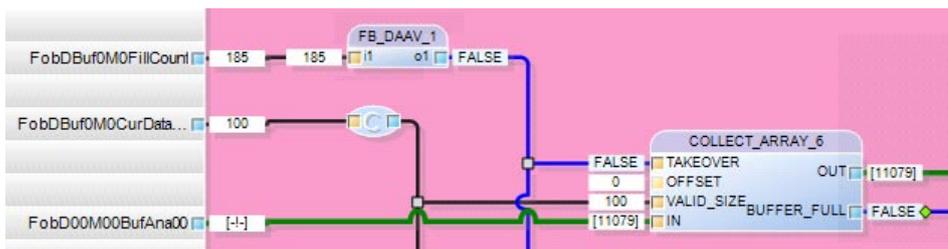
Structured Text

Set breakpoint Delete breakpoint Continue

```

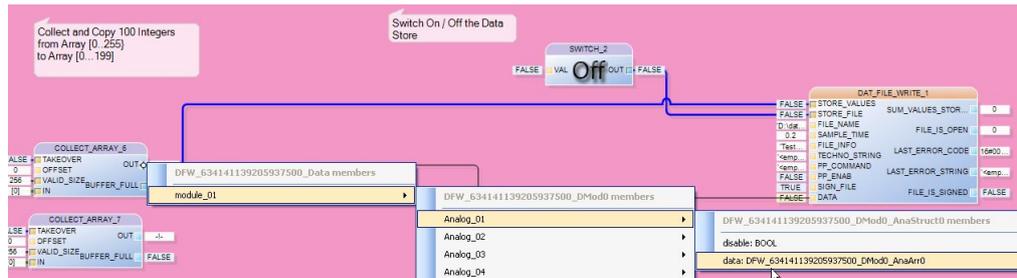
1      (* Set Output TRUE, whenever the input values changes. *)
2      o1 := (i1<>v1); v1 := i1;
```

- Connect the blocks as illustrated in the following screenshot.



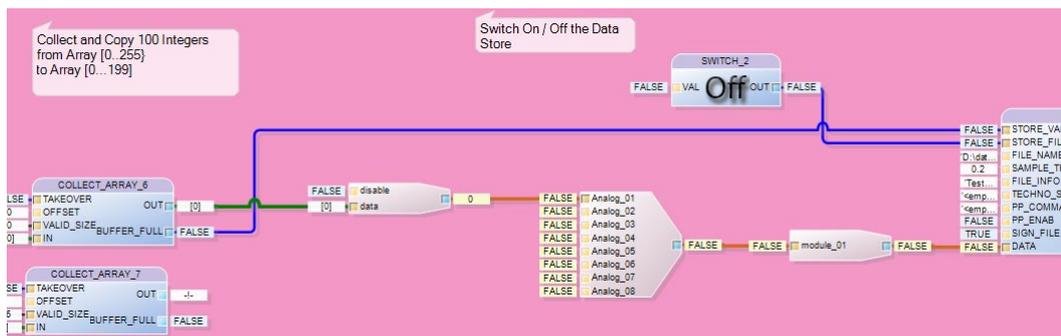
16.2.2.4 Step 4: Transfer the data to DAT_FILE_WRITE

- Create a SWITCH block and connect its output with DAT_FILE_WRITE.STORE_FILES.
 - Connect the COLLECT_ARRAY.BUFFER_FULL with DAT_FILE_WRITE.STORE_VALUES.
 - Connect the output COLLECT_ARRAY.OUT with DAT_FILE_WRITE.DATA.
- ibaLogic pops up the selection menus for the joiner.
Select here the "module_01 - Analog_01 - ... AnaArr0".



Result

The data is transferred to DAT_FILE_WRITE.



16.2.2.5 Step 5: Wiring (Connecting) the remaining inputs

- Copy one COLLECT_ARRAY block for each analog input.
- Connect all COLLECT_ARRAY.TAKEOVER with the DataAvailable signal (FB_DAAV.o1).
- Connect all COLLECT_ARRAY.VALID_SIZE with the output of the converter to "... CurDataSize".
- Connect each COLLECT_ARRAY.IN with the respective buffered analog input "... BufAnann".
- Connect each COLLECT_ARRAY.OUT with the respective connector "Analog_nn" of the joiner.

17 Naming conventions

A name is a string of alphabets, digits and an underscore. The following rules are applicable:

- ❑ Capital and small letters are not relevant, for example, ABCD and abcd are identical.
- ❑ Names must begin with an alphabet or an underscore. A name cannot begin with a digit.
- ❑ Underscores are significant in the names, for example, A_BCD and AB_CD are different names (in contrast to this in number constants).
- ❑ Underscores at the end of a name are not permissible, e. g. ABCD_
- ❑ Multiple underscores are not allowed, e. g. AB__CD
- ❑ Keywords, e. g. for and if, are not allowed.

Peculiarities with ibaLogic:

Names having only one alphabet are not allowed.



Note

These rules, in general, are applicable to ibaLogic, and even beyond function blocks, e. g. for the names of OTCs, IPCs, input and output signals, function block names etc.

18 Data types

18.1 Standard data types

ibaLogic supports the following elementary data types:

Type	Range (Min.)	Range (Max.)	Explanation
BOOL	0 (FALSE)	1 (TRUE)	
BYTE	16#00	16#FF	8-bit
WORD	16#0000	16#FFFF	16-bit
DWORD	16#00000000	16#FFFFFFFF	32-bit word
SINT	-128	127	8-bit signed integer
USINT	0	255	8-bit unsigned integer
INT	-32768	32767	16-bit signed integer
UINT	0	65535	16-bit unsigned integer
DINT	-2147483648	2147483647	32-bit signed integer
UDINT	0	4294967295	32-bit unsigned integer
REAL	1.175494351 e-38	3.402823466 e+38	Floating point, single accuracy, 32 bit
LREAL	2.225073858 ... e-308	1.797693134862 ... e+308	Floating point, double accuracy, 64 bit
TIME	-2147483648ms -24d_20h_31m_23s_648ms	2147483648ms 24d_20h_31m_23s_648ms	Time, mapped internally as DINT with 1 ms resolution per increment
STRING	0	250 characters (249 for the user, on account of NULL flag at the end)	String with number of characters including the end flag (NULL)

18.2 Derived data types

Type	Explanation
DIRECT_DERIVED	Elementary data types with a fixed value (Constants)
SUBRANGE	Integer data types with a limited range of values
STRING_DERIVED	String having a fixed value and length

18.3 Generic data types

Type	Explanation
ENUM	Enumerator type, names are defined instead of values.
ARRAY	Structure, consisting of any sequence of one of the a.m. data types (with the exception of the string that already represents an array); maximum four-dimensional Maximum number of elements: 32767
STRUCT	Structure, consisting of any sequence of the data types mentioned above Maximum number of elements: 1048576



Important Note

Internally, the memory is limited to an internal size of 63 KB for each level. Thus, for example, the memory requirement of all input and output variables of a function block should not exceed this size.

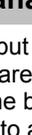
For more information, please refer to "Performance Limits, Page 240".

19 Standard Function Blocks

The Appendix contains a tabular overview of all functions and function blocks that are available in ibaLogic-V4.

19.1 Table interpretation

This section provides tips and instructions on interpreting the tabular overview.

Column	Explanation
Input data type	The input data type columns list the data types permissible for each connector. There are blocks for whose connectors the data types are not defined right from the beginning, but whose data type is defined only when a connection is drawn to another connector.
Block design	
 Green	Functions or function blocks have been defined in conformity with the IEC 61131-3 standard.
 Yellow	Functions or function blocks have been defined by iba AG.
	This block is expandable, i. e. you can change the number of inputs. Open the block by double clicking on it and modify the "Number of inputs".
Output data type	The output data type columns list the data types permissible for each connector. There are blocks for whose connectors the data types are not defined right from the beginning, but whose data type is defined only when a connection is drawn to another connector.
Explanation, example, ST syntax	There is a note provided for each function regarding whether and how you can call up the function within ST. You can also clone functions as multiple lines of ST code. This, however, is not executed.

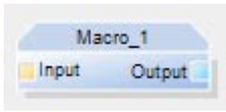
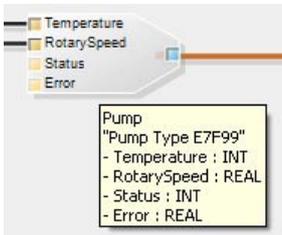
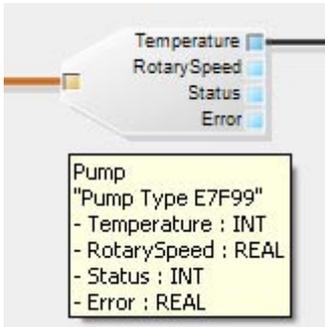
19.2 Data types

The data type "ANY" with the following variants is displayed for the "untyped" connectors:

Data type "ANY" with "untyped" connectors	Explanation
Any_Int	All integer types (SINT, INT, DINT, USINT, UINT and UDINT)
Any_Real	All real types (REAL, LREAL)
Any_Num	All numerical data types (all integers and real values)
Any_Magnitude	All numerical data types and the TIME type
Any_Bit	All bit-oriented types (BOOL, BYTE, WORD and DWORD)
Any_String	STRING data type
Any_Elementary	All elementary types (Integers, real values, TIME and STRING)
Any_Derived	All elementary data types, arrays and structures
Any	Any data type

19.3 Block type with function diagram display

Functions, function blocks and macro blocks are displayed in the design area as follows:

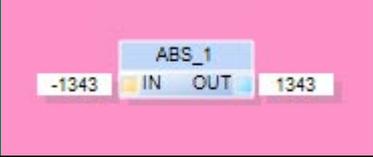
Block type with function diagram display	Explanation
<p>Function</p> 	<p>You can recognize a function from the corners.</p>
<p>Function block</p> 	<p>You can recognize a function block from the rounded corners.</p>
<p>Macro block</p> 	<p>You can recognize a macro block from the flattened corners.</p>
<p>Automatic type converter</p> 	<p>You can recognize a type converter from the letter "C" in the icon.</p>
<p>Automatically generated structure joiner</p> 	<p>Automatically generated structure joiner This is generated automatically as soon as you try to connect a single parameter with a structure.</p>
<p>Automatically generated structure splitter</p> 	<p>Automatically generated structure splitter This is generated automatically as soon as you try to connect a single parameter with a structure.</p>

19.4 Analytical Functions

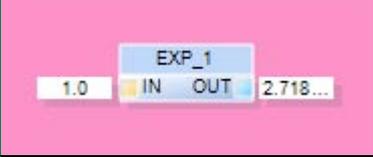
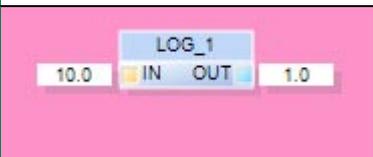
Input data type	Block design	Output data type	Explanation, example, ST syntax
Real Real Bool		Real	<p>DERIVATIVE: Derivative of a value based on the time</p> <p>The output value "OUT" is the derivative of the input value, "VALUE", multiplied with a factor, "FACTOR", of the time dimension. The output is reset with the input "RESET" = "TRUE".</p> <p>Implementation: $OUT := (VALUE_n - VALUE_{n-1}) * FACTOR$</p> <p>ST: cannot be called up</p>
Real Real Bool		Real	<p>INTEGRAL: Integration of the value over time</p> <p>The output value "OUT" is the integral of the input value, "VALUE", multiplied with a factor "FACTOR", of the time dimension. The output is reset with the input, "RESET" = "TRUE".</p> <p>Implementation: $OUT_n := OUT_{n-1} + (VALUE_n * FACTOR);$</p> <p>ST: cannot be called up</p>
Dint Real		Dint Bool Real	<p>MOVING_AVERAGE: Moving average value</p> <p>The input value "COUNT" defines a number of values (= samples) that are considered for average calculation of the value "VALUE". The output value "SIZE", indicates the number of values used for the calculation of the average value. The output "FULL", is "TRUE" if the number of values (samples) specified has been reached. The output value "AVERAGE" yields the cumulative average value. $AVERAGE := (\text{sum of the last SIZE values}) / SIZE.$</p> <p>The average calculation of the value is performed continuously. The number of samples can be modified whenever required.</p> <p>ST: cannot be called up</p>

19.5 Arithmetical Functions

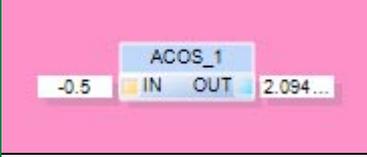
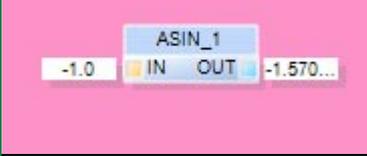
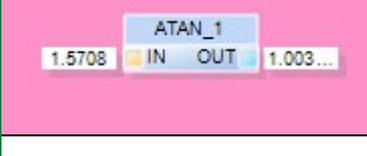
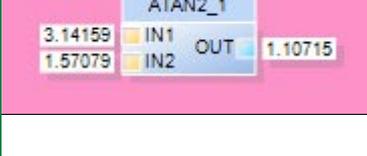
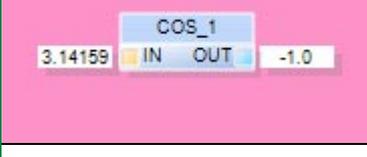
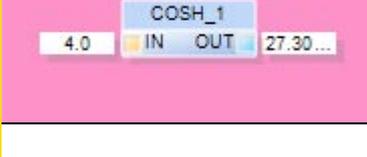
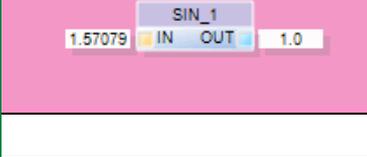
19.5.1 General

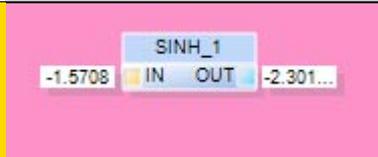
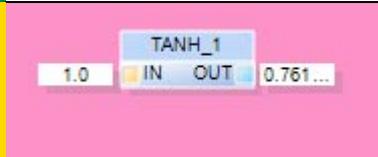
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Num		Any_Num	<p>ABS: Absolute value</p> <p>Example: +1343 = abs(-1343);</p> <p>ST: OUT := abs (IN) ;</p>
Any_Real		Any_Real	<p>SQRT: Square root</p> <p>Example: +3.0 = sqrt(9.0);</p> <p>ST: OUT := sqrt (IN) ;</p>

19.5.2 Logarithmic

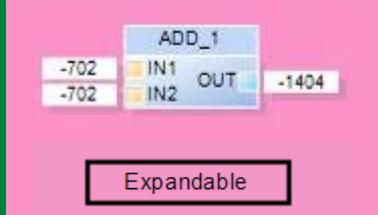
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Real		Any_Real	<p>EXP: Natural exponent to the base e</p> <p>Result: = e^{arg};</p> <p>Examples: 2.71828 = exp(1.0); 0.13533 = exp(-2.0);</p> <p>ST: OUT := exp (IN) ;</p>
Any_Real		Any_Real	<p>LN: Natural logarithm</p> <p>Example: +1.0 = ln(2.71828);</p> <p>ST: OUT := ln (IN) ;</p>
Any_Real		Any_Real	<p>LOG: Logarithm to the base 10</p> <p>Example: +1.0 = log(10.0);</p> <p>ST: OUT := log (IN) ;</p>

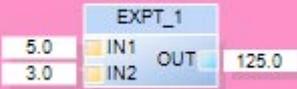
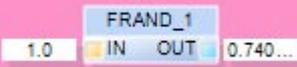
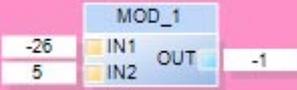
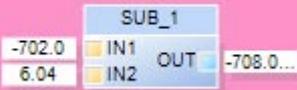
19.5.3 Trigonometric

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Real		Any_Real	<p>ACOS: Arc cosine</p> <p>Example: 1.57079 = acos(0.0);</p> <p>ST: OUT := acos(IN);</p>
Any_Real		Any_Real	<p>ASIN: Arc sine</p> <p>Example: -1.57079 = asin(-1.0);</p> <p>ST: OUT := asin(IN);</p>
Any_Real		Any_Real	<p>ATAN: Arc tan</p> <p>Example: 1.0000 = atan($\pi/2.0$);</p> <p>ST: OUT := atan(IN);</p>
Any_Real Any_Real		Any_Real	<p>ATAN2: Arc tan</p> <p>Example: 1.1071 = atan2_real(p,p/2.0);</p> <p>ST: Different calls for the REAL and LREAL data types OUT := atan2_real(IN1, IN2); OUT := atan2_lreal(IN1, IN2);</p>
Any_Real		Any_Real	<p>COS: Cosine</p> <p>Example: -1.0000 = cos(π);</p> <p>ST: OUT := cos(IN);</p>
Any_Real		Any_Real	<p>COSH: Hyperbolic cosine</p> <p>Example: +27.3082 = cosh_real(4.0);</p> <p>ST: Different calls for the REAL and LREAL data types OUT := cosh_real(IN); OUT := cosh_lreal(IN);</p>
Any_Real		Any_Real	<p>SIN: Sine</p> <p>Example: 1.0 = sin($\pi/2$);</p> <p>ST: OUT := sin(IN);</p>

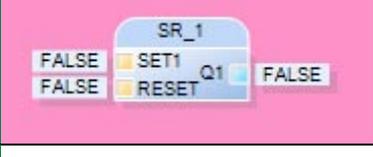
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Real		Any_Real	<p>SINH: Hyperbolic sine</p> <p>Example: -2.3013 = sinh_real(-π/2.0);</p> <p>ST: Different calls for the REAL and LREAL data types OUT:=sinh_real(IN); OUT:=sinh_lreal(IN);</p>
Any_Real		Any_Real	<p>TAN: Tangent</p> <p>Example: 0.648 = tan(10.0);</p> <p>ST: OUT:=tan(IN);</p>
Any_Real		Any_Real	<p>TANH: Hyperbolic tangent</p> <p>Example: 0.76159 = tanh_real(1.0);</p> <p>ST: Different calls for the REAL and LREAL data types OUT:=tanh_real(IN); OUT:=tanh_lreal(IN);</p>

19.5.4 Miscellaneous

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Magnitude Any_Magnitude		Any_Magnitude	<p>ADD: Addition</p> <p>Example: -1404 = -702 + -702;</p> <p>ST: Use operator: OUT:= IN1 + IN2 + ... + INn;</p>
Any_Num Any_Num		Any_Num	<p>DIV: Division</p> <p>Example: -215.3 = -702.0 / 3.26;</p> <p>ST: Use operator: OUT:= IN1 / IN2;</p> <p>Attention: If the divisor IN2 = 0, the result is set to 0 and an error message "Division by Zero" is output cyclically in the event window.</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Real Any_Num		Any_Real	<p>EXPT: General exponent to the base (IN2)</p> <p>Result: $= \text{arg}1^{\text{arg}2}$;</p> <p>Examples: $125.0 = \text{expt}(5.0, 3.0)$; $4.0 = 16.0 ** 0.5$;</p> <p>ST: <code>OUT := expt(IN1, IN2);</code> or <code>OUT := IN1 ** IN2;</code></p>
Any_Real		Any_Real	<p>FRAND: Random number in the range {0 ... arg}</p> <p>Example: $+0.07116 = \text{frand_real}(1.00)$; $+2.92457 = \text{frand_lreal}(6.00)$;</p> <p>ST: Different calls for the REAL and LREAL data types <code>OUT := frand_real(IN);</code> <code>OUT := frand_lreal(IN);</code></p>
Any_Int Any_Int		Any_Int	<p>MOD: Division remainder (Modulo)</p> <p>Example: $-1 = -26 \text{ mod } 5$;</p> <p>ST: Use operator: <code>OUT := IN1 mod IN2;</code></p>
Any_Num Any_Num		Any_Num	<p>MUL: Multiplication</p> <p>Example: $15.0 = 5.0 * 3.0$; $4 = 2 * 2$;</p> <p>ST: Use operator: <code>OUT := IN1 * IN2 * ... * INn;</code></p>
Any_Magnitude Any_Magnitude		Any_Magnitude	<p>SUB: Subtraction</p> <p>Example: $-708.04 = -702 - 6.04$;</p> <p>ST: Use operator: <code>OUT := IN1 - IN2;</code></p>

19.6 Bistable

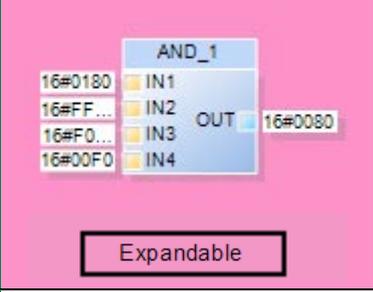
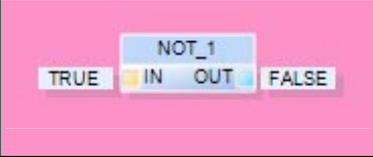
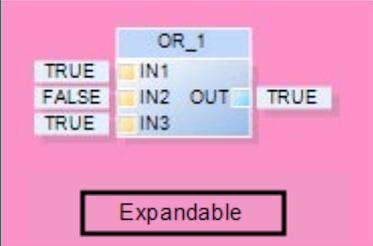
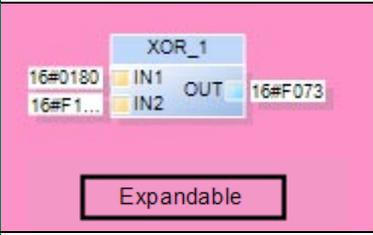
Input data type	Block design	Output data type	Explanation, example, ST syntax																		
Bool Bool		Bool	<p>RS: RS flip-flop (static binary value store) R-dominant</p> <p>Truth table:</p> <table border="1"> <thead> <tr> <th colspan="2">Input values</th> <th>Output</th> </tr> <tr> <th>SET</th> <th>RESET1</th> <th>Q1</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> <p>ST: cannot be called up</p>	Input values		Output	SET	RESET1	Q1	0	0	Q1	0	1	0	1	0	1	1	1	0
Input values		Output																			
SET	RESET1	Q1																			
0	0	Q1																			
0	1	0																			
1	0	1																			
1	1	0																			
Bool Bool		Bool	<p>SR: SR flip-flop (static binary value store) S-dominant</p> <p>Truth table:</p> <table border="1"> <thead> <tr> <th colspan="2">Input values</th> <th>Output</th> </tr> <tr> <th>SET</th> <th>RESET1</th> <th>Q1</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Q1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>ST: cannot be called up</p>	Input values		Output	SET	RESET1	Q1	0	0	Q1	0	1	0	1	0	1	1	1	1
Input values		Output																			
SET	RESET1	Q1																			
0	0	Q1																			
0	1	0																			
1	0	1																			
1	1	1																			

19.7 Bit String

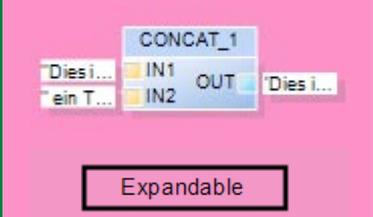
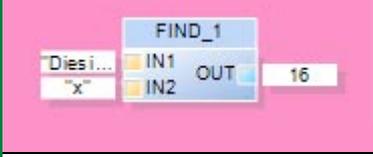
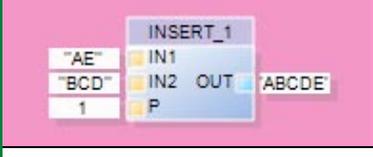
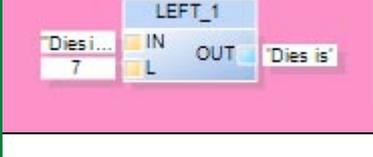
19.7.1 Bit shift

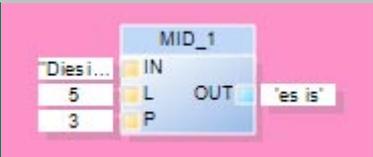
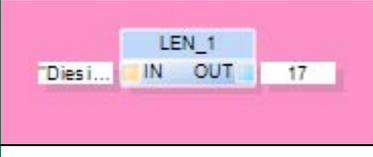
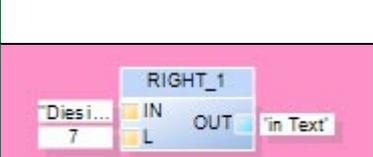
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Bit Uint		Any_Bit	<p>ROL: Rotate arg1 left by arg2 Bits</p> <p>Examples: 16#F50000C2 =rol(16#C2F50000,8); 16#45678123 =rol(16#12345678,12);</p> <p>ST: OUT := rol(IN1, IN2);</p>
Any_Bit Uint		Any_Bit	<p>ROR: Rotate arg1 right by arg2 bits</p> <p>Examples: 16#F00000C2 = ror(16#C2F,4); 16#F500000C = ror(16#CF5,8);</p> <p>ST: OUT := ror(IN1, IN2);</p>
Any_Bit Uint		Any_Bit	<p>SHL: Shift IN1 left by IN2 bits and fill up zeros on the right</p> <p>Example: 16#0D90 = shl(16#00D9,4);</p> <p>ST: OUT := shl(IN1, IN2);</p>
Any_Bit Uint		Any_Bit	<p>SHR: Shift IN1 right by IN2 bits and fill up zeroes on the left</p> <p>Examples: 16#000C = shr(16#0180,5); 16#00D9 = shr(16#0D90,4);</p> <p>ST: OUT := shr(IN1, IN2);</p>

19.7.2 Bitwise_Boolean

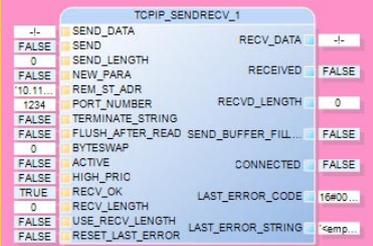
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Bit Any_Bit Any_Bit Any_Bit		Any_Bit	<p>AND: Logical AND combination</p> <p>Example: <code>16#80 = and(16#0180, 16#FFF0, 16#F0F0, 16#00F0);</code></p> <p>ST: Use operator: <code>OUT := IN1 AND IN2 ... AND INn;</code></p>
Any_Bit		Any_Bit	<p>NOT: Logical NOT function</p> <p>Examples: <code>FALSE = not(TRUE);</code> <code>16#FE7F = not(16#0180);</code></p> <p>ST: <code>OUT := not IN;</code> or <code>OUT := not (IN);</code></p>
Any_Bit Any_Bit Any_Bit		Any_Bit	<p>OR: Logical OR combination</p> <p>Examples: <code>1 = or(1, 0, 1);</code> <code>16#F3 = or(16#F0, 16#03);</code></p> <p>ST: Use operator: <code>OUT := IN1 or IN2 or ... INn;</code></p>
Any_Bit Any_Bit		Any_Bit	<p>XOR: Logical XOR combination.</p> <p>Examples: <code>FALSE = xor(TRUE, TRUE);</code> <code>16#F073 = xor(16#0180, 16#F1F3);</code></p> <p>ST: Use operator: <code>OUT := IN1 xor IN2 xor ... INn;</code></p>

19.8 Character String

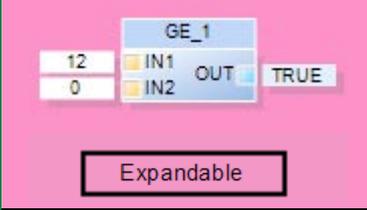
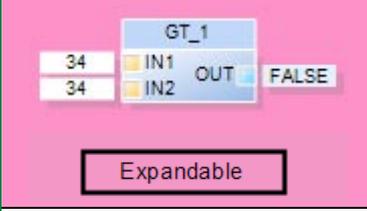
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_String Any_String		Any_String	<p>CONCAT: Combining (joining) sub-strings</p> <p>Examples: 'This is a text'= concat('This is', ' a text')</p> <p>ST: OUT:=concat (IN1, IN2, ..., INn) ;</p>
Any_String UInt UInt		Any_String	<p>DELETE: Delete L characters of a string from (including) position P. The first character has the position 1.</p> <p>Examples: 'This text'= delete('This is a text',8,5); 'DE' = delete('ABCDE',3,1);</p> <p>ST: OUT:=delete (IN, L, P) ;</p>
Any_String Any_String		Int	<p>FIND: Search for the first match of character IN2 in string IN1. If the character is not found, the result = 0.</p> <p>Examples: 16 = find('This is a text', 'x'); 1 = find('This is a text', 'D');</p> <p>ST: OUT:=find (IN1, IN2) ;</p>
Any_String Any_String UInt		Any_String	<p>INSERT: Inserting string IN2 in string IN1 after the position P. If P=0, IN2 is inserted at the beginning.</p> <p>Examples: 'ABCDE' = insert('AE','BCD', 1); 'xABC' = insert('ABC','x',0);</p> <p>ST: OUT:=insert (IN1, IN2, P) ;</p>
Any_String UInt		Any_String	<p>LEFT: The left part of a string IN having length L</p> <p>Example: 'This is'= left('This is a text',7);</p> <p>ST: OUT:=left (IN, L) ;</p>

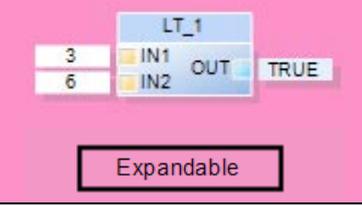
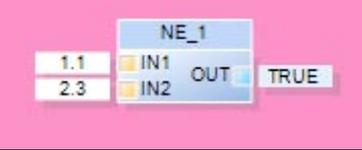
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_String Uint Uint		Any_String	MID: Section of a string IN having length L from and including position P. Example: 'is is' = mid('This is a text',5,3); ST: OUT:=mid (IN, L, P) ;
Any_String		Int	LEN: length of a string (without termination characters) Examples: 17= len('This is a text'); 4= len('text'); ST: OUT:=len (IN) ;
Any_String Any_String Uint Uint		Any_String	REPLACE: Replace L characters of the string IN by IN2 starting from and including position P Example: 'ABXE' = replace('ABCDE','X', 2,3); ST: OUT:= replace (IN1, IN2, L, P) ;
Any_String Uint		Any_String	RIGHT: The right part of a string having length L Example: 'a text'= right('This is a text',6); ST: OUT:= right (IN, L) ;

19.9 Communication

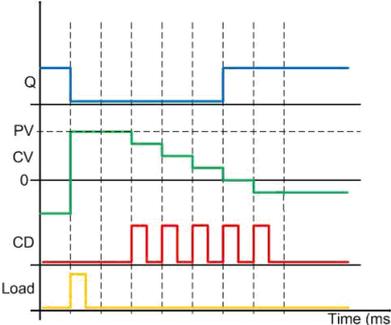
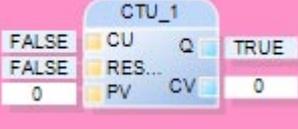
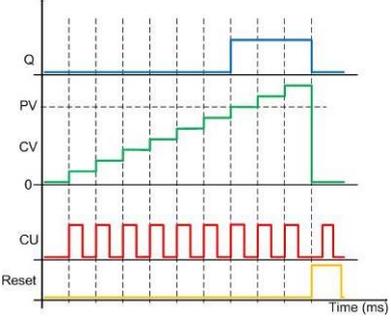
Input data type	Block design	Output data type	Explanation, example, ST syntax
Any Bool Udint Bool String Udint Bool Bool Bool Int Bool Bool Bool Udint Bool Bool		Any Bool Udint Bool Bool Dword String	TCPIP_SendRecv: Transmission and reception of data via TCP/IP. The data here is raw data that is sent via TCP/IP. In this manner, all native TCP/IP protocols can be recreated. You will find a detailed description in "TCPIP_SENDRECV, Page 104". ST: cannot be called up within ST

19.10 Comparison

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_ Elementary Any_ Elementary		Bool	<p>EQ: Comparison of equality</p> <p>The result is TRUE if all arguments are identical.</p> <p>Example: FALSE = (15.3 = 18.6 = 15.3)</p> <p>ST: Use operator: OUT := IN1 = IN2 ;</p> <p>Only two arguments are allowed. Implementation with logical combination of multiple comparisons: OUT := (IN1 = IN2) AND (IN1 = IN3) ;</p>
Any_ Elementary Any_ Elementary		Bool	<p>GE: Comparison for greater than or equal to</p> <p>The result is TRUE if IN1 is greater than or equal to all other arguments.</p> <p>Example: TRUE = 12 >= 0;</p> <p>ST: Use operator: OUT := IN1 >= IN2 ;</p> <p>Only two arguments are allowed. Implementation with logical combination of multiple comparisons: OUT := (IN1 >= IN2) AND (IN1 >= IN3) ;</p>
Any_ Elementary Any_ Elementary		Bool	<p>GT: Comparison for greater than</p> <p>The result is TRUE if IN1 is greater than all other arguments.</p> <p>Example: FALSE = 34 > 34;</p> <p>ST: Use operator: OUT := IN1 > IN2 ;</p> <p>Only two arguments are allowed. Implementation with logical combination of multiple comparisons: OUT := (IN1 > IN2) AND (IN1 > IN3) ;</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_ Elementary Any_ Elementary		Bool	<p>LE: Comparison for less than or equal to</p> <p>The result is TRUE if IN1 is less than or equal to all other arguments.</p> <p>Example: TRUE = (1.2 <= 1.3 <= 1.5);</p> <p>ST: Use operator: OUT := IN1 <= IN2;</p> <p>Only two arguments are allowed. Implementation with logical combination of multiple comparisons: OUT := (IN1 < IN2) AND (IN1 < IN3);</p>
Any_ Elementary Any_ Elementary		Bool	<p>LT: Comparison for less than</p> <p>The result is TRUE if IN1 is less than all other arguments.</p> <p>Example: TRUE = (3 < 6);</p> <p>ST: Use operator: OUT := IN1 < IN2;</p> <p>Only two arguments are allowed. Implementation with logical combination of multiple comparisons: OUT := (IN1 < IN2) AND (IN1 < IN3);</p>
Any_ Elementary Any_ Elementary		Bool	<p>NE: Comparison of inequality</p> <p>The result is TRUE if IN1 is not equal to IN2.</p> <p>Example: TRUE = ('Text 1' <> 'Text 2');</p> <p>ST: Use operator: OUT := IN1 <> IN2;</p>

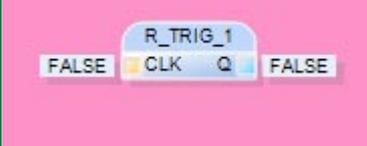
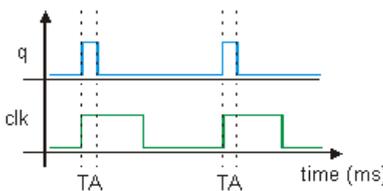
19.11 Counter

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Bool Int		Bool Int	<p>CTD: Count down (Downwards counter)</p> <p>When the counter is set with LOAD = 1 the counter value CV is set to the initial value PV. With each rising edge of CD, the counter value CV is decremented by one. As soon as the counter output is CV ≤ 0, the output is set Q = 1. The CV output runs down to a minimum value of -32,768.</p>  <p>ST: cannot be called up</p>
Bool Bool Int		Bool Int	<p>CTU: Count up (Upwards counter)</p> <p>With each rising edge of CU, the counter value CV is incremented by one. As soon as the counter output CV ≥ count value PV, the output Q is set to "TRUE". When "RESET" = 1, the output Q is set to "FALSE" and the output CV is set to 0. The CV output runs up to a maximum value of 32767.</p>  <p>ST: cannot be called up</p>

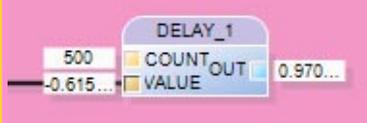
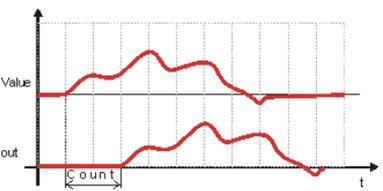
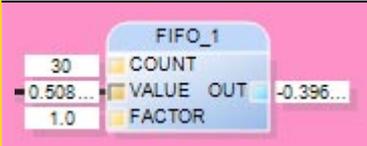
Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Bool Bool Bool Int		Bool Bool Int	<p>CTUD: Counter for counting up and down (Up/Down counter)</p> <p>With each rising edge of CU, the counter value "CV" is incremented by one per sampling time. When the counter output is "CV" >= count value "PV", the output is QU = 1 (Flow diagram see "CTU" FB).</p> <p>When the counter is set with "LOAD" = 1, the counter value "CV" is set to the initial value "PV".</p> <p>With each rising edge of "CD", the counter value "CV" is decremented by one. As soon as the counter output is "CV" <= 0, the output is "QD" = 1 (Flow diagram see "CTD" FB).</p> <p>When the counter is "RESET" = 1, the counter output is set to 0.</p> <p>Range of values for "CV": -32,768 to 32,767</p> <p>ST: cannot be called up</p>

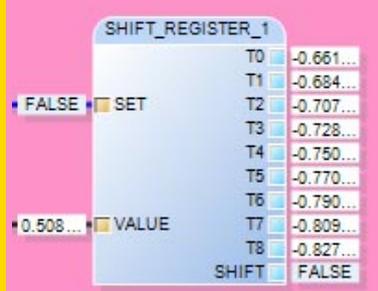
19.12 Edge Detection

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool		Bool	<p>F_TRIG: Detecting falling edges</p> <p>With a falling edge at the input "CLK", the output Q is set to "TRUE" for one task cycle.</p> <p>Start-up behavior: When the input "CLK", is "FALSE" at the time of system start-up, the function block generates a pulse at the output Q = "TRUE" for a period of one cycle.</p> <p>ST: cannot be called up</p>

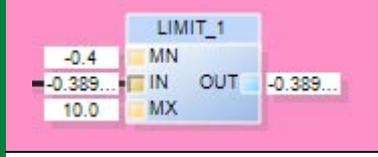
Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool		Bool	<p>R_TRIG: Detecting falling edges</p> <p>With a rising edge at the input "CLK", the output Q is set to "TRUE" for one task cycle.</p> <p>Start-up behavior: When the input; "CLK", is "TRUE" at the time of system start-up, the function block generates a pulse at the output Q = "TRUE" for a period of one cycle.</p>  <p>ST: cannot be called up</p>

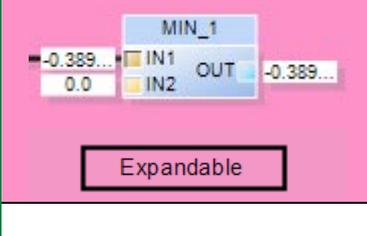
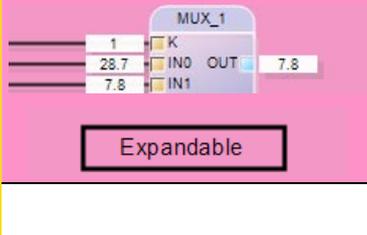
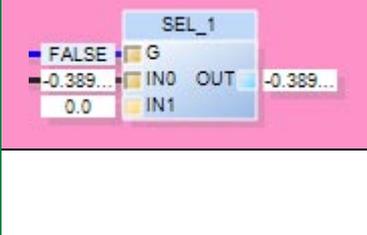
19.13 Register

Input data type	Block design	Output data type	Explanation, example, ST syntax
Dint Any		Any	<p>DELAY: Time delay feature</p> <p>The output value "OUT", follows the input value "VALUE", with a time delay that is specified by the "COUNT" input in number of cycles.</p>  <p>When you use the "ARRAY" data type ("VALUE" and "OUT"), the block is limited on account of memory capacity. If the number of "ARRAY" elements exceeds 64, the range of values of the time delay of 65,536 is reduced accordingly.</p> <p>ST: cannot be called up</p>
Dint Real Real		Real	<p>FIFO: First In First Out - Storage</p> <p>The output value "OUT", follows the input value "VALUE", with a time delay that is specified by the "COUNT" input in number of cycles. In addition, the input value is multiplied with "FACTOR".</p> <p>ST: cannot be called up</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax																		
Any_Magnitude Bool Bool Any_Magnitude		Any_Magnitude	<p>REGISTER: Register memory The block works with the signal state and not with the signal edges.</p> <p>Function table:</p> <table border="1"> <thead> <tr> <th colspan="2">Input values</th> <th>Output</th> </tr> <tr> <th>SET</th> <th>RESET</th> <th>OUT</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>OUT_{n-1}</td> </tr> <tr> <td>0</td> <td>1</td> <td>RESETVALUE</td> </tr> <tr> <td>1</td> <td>0</td> <td>VALUE</td> </tr> <tr> <td>1</td> <td>1</td> <td>VALUE</td> </tr> </tbody> </table> <p>ST: cannot be called up</p>	Input values		Output	SET	RESET	OUT	0	0	OUT _{n-1}	0	1	RESETVALUE	1	0	VALUE	1	1	VALUE
Input values		Output																			
SET	RESET	OUT																			
0	0	OUT _{n-1}																			
0	1	RESETVALUE																			
1	0	VALUE																			
1	1	VALUE																			
Bool Real		Real Real Real Real Real Real Real Real Real Real Bool	<p>SHIFT_REGISTER: Shift register As long as the input "SET" = "TRUE", the input value, "VALUE" is shifted by an output T_i in every task cycle.</p> <p>Shift, if "SET" = "TRUE"</p> <p>T₀: = VALUE(T_n) = current cycle T₁: = VALUE(T_{n-1}) = previous cycle T₈: = VALUE(T_{n-8}) = oldest cycle where n = task cycle</p> <p>ST: cannot be called up</p>																		

19.14 Selection

Input data type	Block design	Output data type	Explanation, example, ST syntax
Any_Elementary Any_Elementary Any_Elementary		Any_Elementary	<p>LIMIT: Limit value The input value IN is limited to the limit values MN (min.) and MX (max.).</p> <p>Example: -0.389 = limit(-0.4, -0.389, 10.0); 15.3 = limit(8.9, 17.6, 15.3);</p> <p>ST: OUT := limit(MN, IN, MX);</p>
Any_Elementary Any_Elementary		Any_Elementary	<p>MAX: Maximum value</p> <p>Examples: 0.3 = max(-0.389, 0.3); 12 = max(0, 10, 12, 5);</p> <p>ST: OUT := max(IN1, IN2, ..., INn);</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax						
Any_ Elementary Any_ Elementary		Any_ Elementary	<p>MIN: Minimum value</p> <p>Examples: -0.389 = min(-0.389, 0.3); 0 = min(0, 10, 12, 5);</p> <p>ST: OUT := min(IN1, IN2, ..., INn);</p>						
Dint Any Any		Any	<p>MUX: Multiple selector</p> <p>Expandable selection block for any data types. All selection values have to be of the same data type. "K" = Selector, "IN0..IN63" selection values, "OUT" resulting value.</p> <p>ST: cannot be called up</p>						
Bool Any_ Elementary Any_ Elementary		Any_ Elementary	<p>SEL: Selector</p> <p>Selection (1 out of 2) with binary signal "G"</p> <p>Function table:</p> <table border="1" data-bbox="1077 929 1465 1052"> <thead> <tr> <th>SEL</th> <th>OUT</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>IN0</td> </tr> <tr> <td>1</td> <td>IN1</td> </tr> </tbody> </table> <p>Example: -0.389 = sel(FALSE, -0.389, 0);</p> <p>ST: Different calls for "REAL" and "INT" data types, and other data types are not possible.</p> <p>ST: OUT := sel_real(G, IN0, IN1); OUT := sel_int(G, IN0, IN2);</p>	SEL	OUT	0	IN0	1	IN1
SEL	OUT								
0	IN0								
1	IN1								

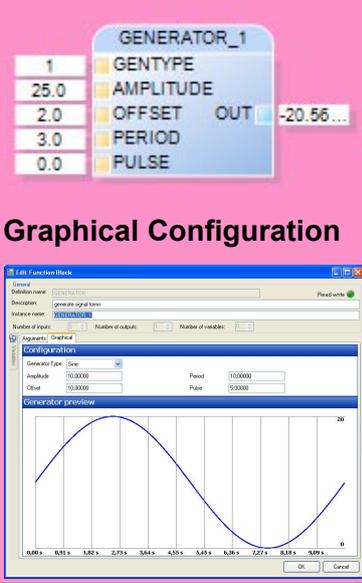
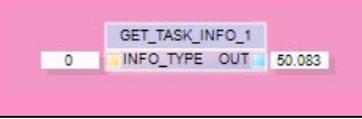
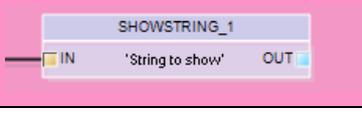
19.15 Signal Processing

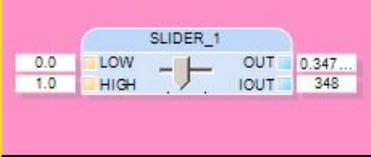
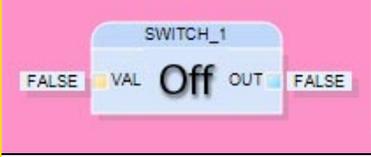
Input data type	Block design	Output data type	Explanation, example, ST syntax
<p>One-dimensional real array having a depth of 2^n</p> <p>Bool</p>		<p>One one-dimensional real array each having a depth of 2^{n-1}</p>	<p>CRFFT: Fast Fourier Transformation with an imaginary component</p> <p>There must be a one-dimensional array of "REAL" type and having 2^{**n} elements at the input "IN". The output is then always two arrays of the same type having the length $2^{**}(n-1)$.</p> <p>Example: <code>IN ← ARRAY [0...2047]</code> <code>OF REAL</code> <code>ROUT → ARRAY [0...1023]</code> <code>OF REAL</code> <code>IOUT → ARRAY [0...1023]</code> <code>OF REAL</code></p> <p>The FFT evaluation is enabled when the input, "TRIGGER" = "TRUE". It is only then that the block requires computing time!</p> <p>This function block delivers the real part at the output "ROUT" and the imaginary part at the output, "IOUT", of an FFT evaluation.</p> <p>Evaluation mode: Absolute amplitude, all values in the array have the same weight (Rectangular window).</p> <p>ST: cannot be called up</p>
<p>One-dimensional real array having a depth of 2^n</p> <p>Bool</p>		<p>One one-dimensional real array each having a depth of 2^{n-1}</p>	<p>RFFT: Fast Fourier Transformation</p> <p>There must be a one-dimensional array of "REAL" type and having 2^{**n} elements at the input "IN". The output is then always two arrays of the same type having the length $2^{**}(n-1)$.</p> <p>Example: <code>IN ← ARRAY [0 ... 2047] OF REAL</code> <code>OUT → ARRAY [0 ... 1023] OF REAL</code></p> <p>The FFT evaluation is enabled when the input, "TRIGGER" = "TRUE". Here, too, it is only then that the block requires computing time!</p> <p>This function block delivers the result of an FFT at the output according to the evaluation mode: Absolute amplitude, all values in the array have the same weight (Rectangular window).</p> <p>ST: cannot be called up</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Real Real Any_ Derived		Any_ Derived	<p>SCALE_ARRAY: As long as the input "DOCONVERT" = "TRUE", each element in the input array "IN", is multiplied with "SCALE" and added with the value at the "OFFSET" input.</p> <p>You then have the scaled array available at the output "OUT".</p> <p>ST: cannot be called up</p>

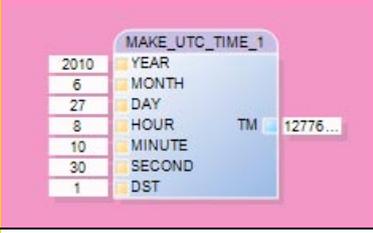
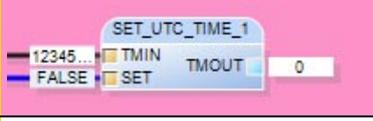
19.16 Specials

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Bool String Lreal Lreal Lreal Lreal Lreal Lreal Lreal Lreal		Dint Usint Dword String Bool	<p>DAT FILE WRITE: You can use this block to record signals directly in ibaLogic for subsequent analysis using the ibaAnalyzer.</p> <p>You will find a detailed description in "DAT_FILE_WRITE (DFW Function Block), Page 96".</p> <p>ST: cannot be called up</p>
Bool		Real Real Real Real Udint Real	<p>EVALTIMES: Output of the evaluation data</p> <p>EVAL_DELTA_TIME = current cycle time of the task (in ms)</p> <p>MAX_DELTA_TIME = max. cycle time of the task since the previous start (in ms)</p> <p>MIN_DELTA_TIME = minimum cycle time</p> <p>EVAL_TIME = time elapsed since the previous start (in ms)</p> <p>EVAL_TIME_TICK = time elapsed since the previous start in µs</p> <p>TASK_DURATION = Evaluation time of the current task.</p> <p>ST: cannot be called up</p>
Bool Bool String Lreal Lreal Lreal Lreal Lreal Lreal Lreal Lreal		Lreal Array [0..8] of Lreal String Int String	<p>FUZZY_CONTROLLER: Controller block using fuzzy logic.</p> <p>You will find a short description in "FUZZY_CONTROLLER, Page 119".</p> <p>ST: cannot be called up</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
<p>Int Real Real Real Real</p>		<p>Real</p>	<p>GENERATOR: Function generator for sinusoidal, rectangular (square wave) and triangular (saw tooth) signals.</p> <p>"GENTYPE" = 1 for sinusoidal, 2 for rectangular (square wave) and 3 for triangular (saw tooth) signal</p> <p>"AMPLITUDE" = Amplitude value; there is only one value that is evaluated symmetrically to the X-axis, i. e. it applies to both positive and negative values.</p> <p>"OFFSET" = Specification of the offset (position of the X-axis); if you desire a trend graph in which the value is non-negative, you must choose the offset at least as large as the amplitude.</p> <p>"PERIOD" = Specification of the time period in seconds</p> <p>"PULSE" (Pulse width) = specification of the time for the first pulse in seconds; it is not used for sinusoidal waveforms. The value should not be greater than the time period. For a symmetric signal, Pulse = Period / 2</p> <p>The specialty of this block is that you also have the option of configuring the signals graphically. You can select this interface by double clicking on the block. You can these use the mouse to set values in the graphical display.</p> <p>ST: cannot be called up</p>
<p>Int</p>		<p>Real</p>	<p>GET_TASK_INFO: Function to extract task information corresponding to the "INFO_TYPE" parameter.</p> <p>"INFO_TYPE" = 0: EvalDeltaTime, 1: EvalTime, 2: LastTaskDuration</p> <p>ST: cannot be called up</p>
<p>String</p>		<p>String</p>	<p>SHOWSTRING: Display element for displaying strings.</p> <p>ST: cannot be called up</p>

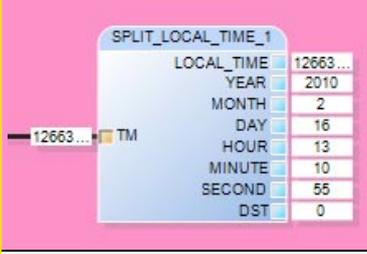
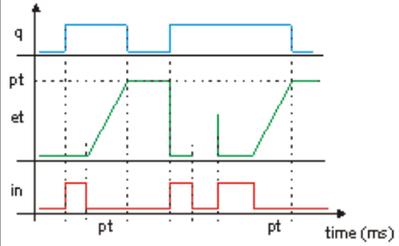
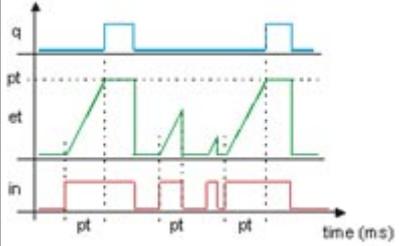
Input data type	Block design	Output data type	Explanation, example, ST syntax															
Real Real		Real Dint	<p>SLIDER: slide controller</p> <p>Depending on the position of the slider, this function block delivers a value at its output "OUT", where the value lies between the limits of the input specifications (Minimum and maximum value). The inputs are preset by default to 0 and 1, but can be modified as required. (Double click on the module and adjust the default values)</p> <p>The output "IOUT", delivers the relative positional value of the slider in steps of one-thousandth (0 ... 1000).</p> <p>The outputs are set only when the mouse button releases the slider.</p> <p>If the slider pointer is marked, it can also be moved using the cursor keys → und ← .</p> <p>ST: cannot be called up</p>															
Bool		Bool	<p>SWITCH: Switch</p> <p>You have to click with the left mouse button on the "OFF" icon to switch on or off ("toggle").</p> <p>In conjunction with the input, you have an "OR" function between the switch position and the input.</p> <p>Truth table:</p> <table border="1" data-bbox="1077 1187 1468 1400"> <thead> <tr> <th>SWITCH</th> <th>VAL</th> <th>OUT</th> </tr> </thead> <tbody> <tr> <td>ON</td> <td>FALSE</td> <td>TRUE</td> </tr> <tr> <td>ON</td> <td>TRUE</td> <td>TRUE</td> </tr> <tr> <td>OFF</td> <td>FALSE</td> <td>FALSE</td> </tr> <tr> <td>OFF</td> <td>TRUE</td> <td>TRUE</td> </tr> </tbody> </table> <p>ST: cannot be called up</p>	SWITCH	VAL	OUT	ON	FALSE	TRUE	ON	TRUE	TRUE	OFF	FALSE	FALSE	OFF	TRUE	TRUE
SWITCH	VAL	OUT																
ON	FALSE	TRUE																
ON	TRUE	TRUE																
OFF	FALSE	FALSE																
OFF	TRUE	TRUE																

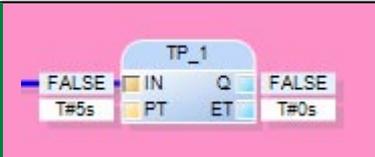
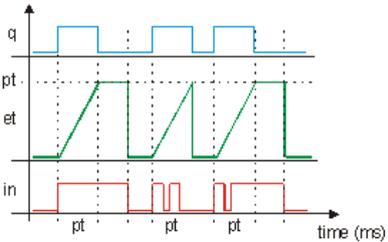
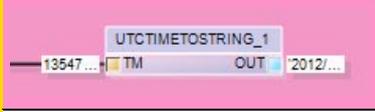
19.17 Timer

Input data type	Block design	Output data type	Explanation, example, ST syntax
Dint Dint Dint Dint Dint Dint Dint Dint		Udint	<p>MAKE_UTC_TIME: Coding the UTC time⁸</p> <p>The function block generates the UTC time at the output "TM" from the input variables "YEAR", "MONTH", "DAY", "HOUR", "MINUTE" and "SECOND".</p> <p>The local timezone⁹ is not taken into consideration. You can specify the Daylight Savings Time at the "DST" input.</p> <p>Example: 27.06.2010/08:10:30 in the timezone GMT+01 → TM = 1277626230</p> <p>ST: cannot be called up</p>
Udint Bool		Udint	<p>SET_UTC_TIME: Set the UTC time</p> <p>The function block set the "UTC System Time" of the ibaLogic platform (Windows PC or PADU-S-IT) to the value at the "TMIN" input when the input "SET" = "TRUE".</p> <p>The local time zone and the Daylight Savings Time (DST) is not taken into consideration.</p> <p>ST: cannot be called up</p>
Udint		Udint Dint Dint Dint Dint Dint Dint Dint	<p>SPLIT_UTC_TIME: Decoding of UTC time in GMT.</p> <p>The function block generates the output variables, "YEAR", "MONTH", "DAY", "HOUR", "MINUTE" and "SECOND" from the UTC time at the TM input.</p> <p>The local time zone is not taken into consideration. The Daylight Savings Time is displayed at the "DST" output.</p> <p>ST: cannot be called up</p>

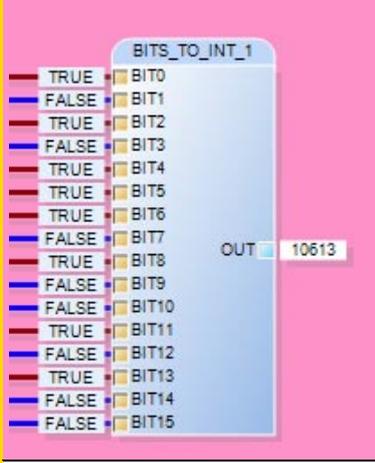
⁸ The UTC time (Universal Time Coordinated) contains the time in seconds since 01.01.1970, 00:00 midnight, related to GMT+00.

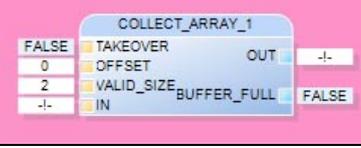
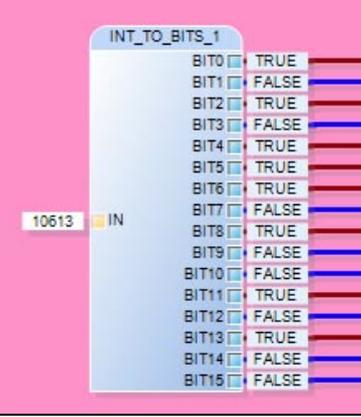
⁹ The information regarding the time zone and summer time (DST) is taken over from the Operating System settings under Windows XP or Windows CE.

Input data type	Block design	Output data type	Explanation, example, ST syntax
Udint		Udint Dint Dint Dint Dint Dint Dint Dint	<p>SPLIT_LOCAL_TIME: Decoding of UTC time in local time.</p> <p>The function block generates the output variables "YEAR", "MONTH", "DAY", "HOUR", "MINUTE" and "SECOND" from the UTC time at the TM input.</p> <p>The local time zone is not taken into consideration. The Daylight Savings Time is displayed at the "DST" output.</p> <p>ST: cannot be called up</p>
Bool Time		Bool Time	<p>TOF: Off delay (Switch off time delay)</p> <p>If the input "IN", is "TRUE", the output "Q", is set to "TRUE" without any time delay. The falling edge at the "IN" input starts the time delay PT. After the delay time has elapsed, the output "Q", is set to "FALSE". The output "Q", remains unchanged if the switch-off time of "IN" is shorter than the time delay. The output "ET", indicates the time that has already elapsed.</p>  <p>ST: cannot be called up</p>
Bool Time		Bool Time	<p>TON: On delay (Switch on time delay)</p> <p>The rising edge at the "IN" input starts the time delay PT. After the delay time has elapsed, the output "Q", is set to "FALSE". A "FALSE" signal at the input "IN", is immediately transferred to the output "Q". The output "Q", is not set if the switch-on time of "IN" is shorter than the time delay. The output "ET", indicates the time that has already elapsed.</p>  <p>ST: cannot be called up</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Time		Bool Time	<p>TP: Timer pulse (Pulse extension)</p> <p>The rising edge at the input "IN", sets the output "Q", for the pulse time PT to "TRUE". The output "Q", cannot be reset during the timer pulse. The output "ET", indicates the time that has already elapsed.</p>  <p>ST: cannot be called up</p>
Udint		String	<p>UTCTIMETOSTRING: Converts UTC time in a formatted string.</p> <p>ST: OUT := UTCTIMETOSTRING (IN) ;</p>

19.18 Type Conversion

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool		Int	<p>BITS_TO_INT: Converts 16 bits to an integer value</p> <p>Example: 10613 = 2#0010_1001_0111_0101, (Bit15 Bit0)</p> <p>ST: cannot be called up</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Bool UInt Uint Any_ Derived		Any_ Derived Bool	<p>COLLECT_ARRAY You can use this block to transfer sections of one array to another.</p> <p>"TAKEOVER": As long as this input is "TRUE", data is transferred to the output array.</p> <p>"OFFSET": This specifies the element in the input array from which data needs to be copied.</p> <p>"VALID_SIZE": Number of elements to be copied.</p> <p>"IN": Input array of any size.</p> <p>"OUT": Output array</p> <p>"BUFFER_FULL": Output array is full, and data has to be read out.</p> <p>ST: cannot be called up</p>
Dword		String String String String String	<p>DWORD_TO_CHAR Conversion of a "DWORD" to four separate characters of "STRING" type</p> <p>16#22645240 = ' @', 'R', 'd', ''</p> <p>ST: cannot be called up within ST</p>
Int		Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool Bool	<p>INT_TO_BITS: Converts an integer value to 16 bits (Inverse function of BITS_TO_INT)</p> <p>ST: cannot be called up</p>

19.18.1 Limiting Converter

These conversion blocks take on a special role, since, prior to the type conversion, they limit the range of values of the input type to the range of values of the output type. The following example illustrates the difference with respect to a standard converter.

Limit converter: `limit:dint_to_int(57700)` delivers the result: 32767

(First, the output value is limited to the range of values (-32768 ... 32767), and then type conversion is carried out).

Standard converter: `dint_to_int(577000)` delivers the result: -12824

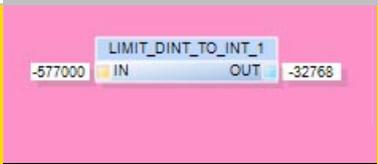
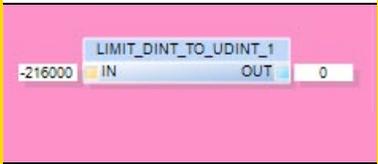
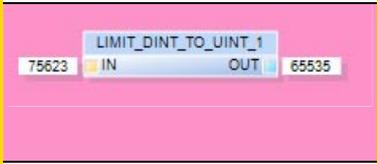
(The 16 lower order bits are considered and these are converted, whereby, naturally, the highest order bit (Bit 15) is interpreted as a sign bit.)

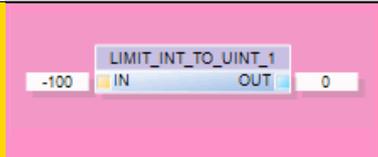
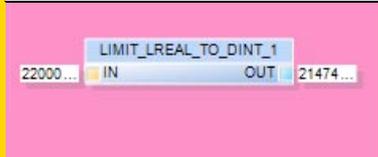
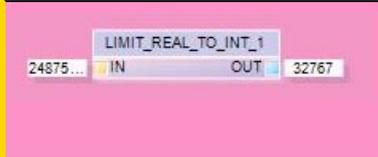
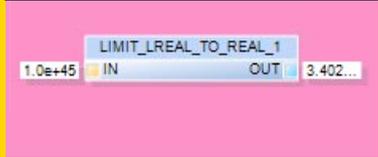
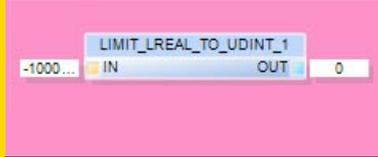
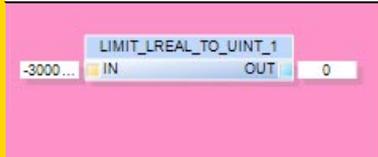
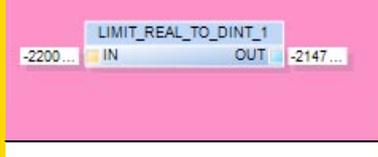
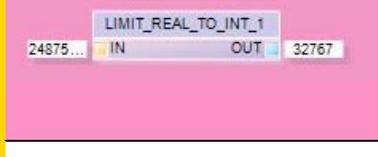
We recommend that you use a limiting converter if the range of values of the target type is smaller than the range of values of the source type. This concerns the following conversions:

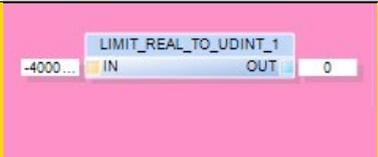
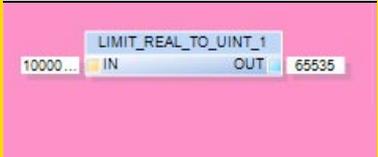
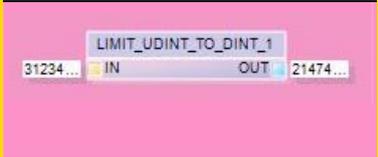
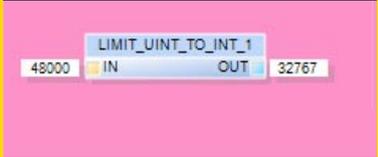
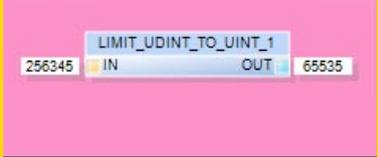
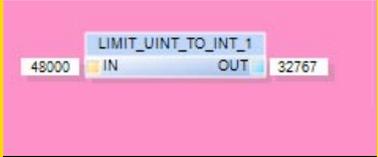
INT	UINT	DINT	UDINT	REAL	LREAL
INT → UINT INT → SINT INT → USINT	UNIT → INT UINT → SINT UINT → USINT	DINT → INT DINT → UDINT DINT → UINT DINT → SINT DINT → USINT	UDINT → DINT UDINT → INT UDINT → UINT UINT → SINT UINT → USINT	REAL → DINT REAL → INT REAL → UDINT REAL → UINT REAL → SINT REAL → USINT	LREAL → DINT LREAL → INT LREAL → REAL LREAL → UDINT LREAL → UINT LREAL → SINT LREAL → USINT

In Structured Text, the limiting converters are provided by the functions:

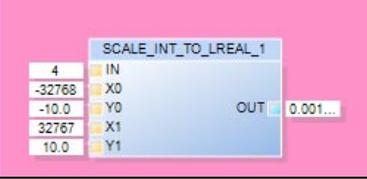
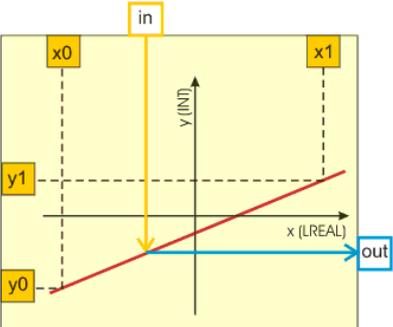
"`limit_source_type_to_target_type`"

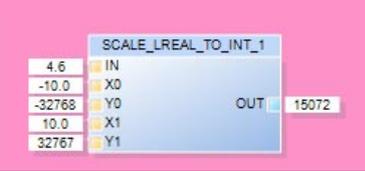
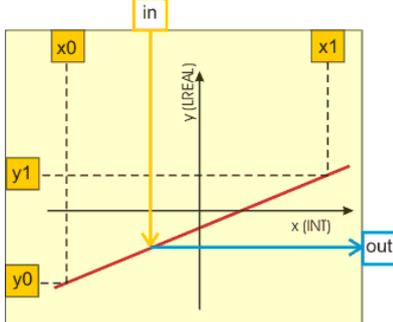
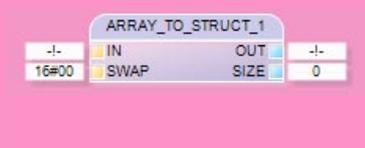
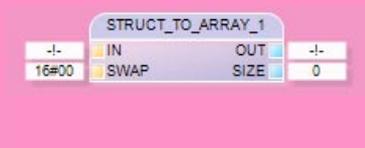
Input data type	Block design	Output data type	Explanation, example, ST syntax
Dint		Int	LIMIT_DINT_TO_INT: Example: -577000 => -32768; ST: OUT:= limit_dint_to_int(IN);
Dint		Udint	LIMIT_DINT_TO_UDINT: Example: -216000 => 0; ST: OUT:= limit_dint_to_udint(IN);
Dint		Uint	LIMIT_DINT_TO_UINT: Example: 75623 => 65535 ST: OUT:= limit_dint_to_uint(IN);

Input data type	Block design	Output data type	Explanation, example, ST syntax
Int		Uint	<p>LIMIT_INT_TO_UINT</p> <p>Example: -100 => 0</p> <p>ST:</p> <pre>OUT:= limit_int_to_uint(IN);</pre>
Lreal		Dint	<p>LIMIT_LREAL_TO_DINT:</p> <p>Example: 2.2 E+09 => 2147483648;</p> <p>ST:</p> <pre>OUT:= limit_lreal_to_dint(IN);</pre>
Lreal		Int	<p>LIMIT_LREAL_TO_INT:</p> <p>Example: 248758.0 => 32767</p> <p>ST:</p> <pre>OUT:= limit_lreal_to_int(IN);</pre>
Lreal		Real	<p>LIMIT_LREAL_TO_REAL</p> <p>Example: 1E+45 => 3.402823466 E+38</p> <p>ST:</p> <pre>OUT:= limit_lreal_to_real(IN);</pre>
Lreal		Udint	<p>LIMIT_LREAL_TO_UDINT:</p> <p>Example:-1 E+12 => 0;</p> <p>ST:</p> <pre>OUT:= limit_lreal_to_udint (IN);</pre>
Lreal		Uint	<p>LIMIT_LREAL_TO_UINT:</p> <p>Example:-3 E+12 => 0;</p> <p>ST:</p> <pre>OUT:= limit_lreal_to_uint(IN);</pre>
Real		Dint	<p>LIMIT_REAL_TO_DINT:</p> <p>Example: -2.2 E+09 => -2147483648;</p> <p>ST:</p> <pre>OUT:= limit_real_to_dint(IN);</pre>
Real		Int	<p>LIMIT_REAL_TO_INT:</p> <p>Example: 248758.0 => 32767;</p> <p>ST:</p> <pre>OUT:= limit_real_to_int(IN);</pre>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Real		Udint	LIMIT_REAL_TO_UDINT: Example: -4000.0 => 0 ST: OUT:= limit_real_to_udint(IN);
Real		Uint	LIMIT_REAL_TO_UINT: Example: 1*E+12 => 65535; ST: OUT:= limit_real_to_uint(IN);
Udint		Dint	LIMIT_UDINT_TO_DINT: Example: 3123456789 => 2147483647; ST: OUT:= limit_udint_to_dint(IN);
Udint		Int	LIMIT_UDINT_T_INT: Example: 558900 => 32767 ST: OUT:= limit_udint_to_int(IN);
Udint		Uint	LIMIT_UDINT_TO_UINT: Example: 256345 => 65535 ST: OUT:= limit_udint_to_uint(IN);
Uint		Int	LIMIT_UINT_TO_INT: Example: 48000 => 32767; ST: OUT:= limit_uint_to_int(IN);

19.18.2 Scaling Converter

Input data type	Block design	Output data type	Explanation, example, ST syntax
Int Int Lreal Int Lreal		Lreal	<p>SCALE_INT_TO_LREAL: This module converts an "INTEGER" value to an "LREAL" value and scales it linearly.</p> <p>Application: Conversion of an analog input (Integer value -32768 ... 32767) to a physical parameter e. g. +/- 10 Volt.</p> <p>IN: Input value (Analog input) X0, X1: Range of values of input value (Int) Y0, Y1: Range of values Target parameter (Lreal)</p> <p>Example: 4 → 0.0013733119 V X0, X1 = -32768 / +32767 Y0, Y1 = -10.0 / + 10.0 IN = 4 OUT = 0.0013733119</p>  <p>Implementation (Type conversions have been omitted for the sake of clarity):</p> <pre> dx := x1-x0; if (dx <> 0.0) then aa := (y1 - y0) / dx; bb := y0 - aa*x0; out = aa*in + bb; end_if; </pre> <p>ST: cannot be called up</p> <p>Important note: Since the integer range of values is not symmetric in principle, a 0 at the input leads to a 0 at the output. Since this always leads to misinterpretations, iba AG recommends that you specify a symmetric range of values for the input (-32,767/+ 32,767).</p>

Input data type	Block design	Output data type	Explanation, example, ST syntax
Lreal Lreal Int Lreal Int		Int	<p>SCALE_LREAL_TO_INT: This module converts an "LREAL" value to an "INTEGER" value and scales it linearly.</p> <p>Application: Conversion of physical parameter (e. g. +/- 10 Volt) to an analog output (Integer value -32,768 ... 32,767)</p> <p>Example: 4.6 V → 15072 X0, X1 = physical range (-/+ 10 V) Y0, Y1 = range of values INT</p>  <p>Implementation (Type conversions have been omitted for the sake of clarity):</p> <pre> dx := x1-x0; if (dx <> 0.0) then aa := (y1 - y0) / dx; bb := y0 - aa*x0; out := aa*in + bb; end_if; </pre> <p>ST: cannot be called up</p>
Any_Array Byte		Any_Struct Int	<p>ARRAY_TO_STRUCT: Creates a structure from any array. SIZE includes the effectively used data size.</p> <p>SWAP sets the byte swap: 16#00 Off, 16#01 depending on data type (AB CDEF ↔ ♦ BA FEDC), 16#02 2 bytes (ABCD ↔ ♦ BADC), 16#04 4 bytes (ABCD ↔ ♦ DCBA).</p> <p>ST: cannot be called up</p>
Any_Struct Byte		Any_Array Int	<p>STRUCT_TO_ARRAY: Creates an array from any structure. (Parameter see ARRAY_TO_STRUCT)</p> <p>ST: cannot be called up</p>

19.18.3 Standard Converter

All standard conversion blocks are combined into one block. As soon as you drag this block and drop it in the design area, a selection dialog box appears with which you can define the input and output data types.

Conversion rules:

- For target type "BOOL", the result is "FALSE" if the input has the value 0, 0.0, 16#0 or T#0ms, otherwise, "TRUE" is output.
- For conversion of real data types to integer data types, the values are converted numerically and rounded up or off in accordance with the rules of arithmetic. The values are not limited.
Example: 82600.0(REAL) → 82600(DINT) → 17064(INT)
- Conversion from "INTEGER" and "WORD" data types takes place using type conversion without changing the bit pattern. Any limiting required is not carried out and the higher orders bits are truncated.
- For real data types to "WORD" data types, the value is first converted to "DINT" numerically and then only data type conversion is done without changing the bit pattern.

If you join two connectors of different data types with one another, a suitable conversion block is automatically created, provided that conversion is possible. For more information, please refer to "Converter, Page 162".

Input data type	Block design	Output data type	Explanation, example, ST syntax
			<p>TYPE TO TYPE:As soon as you drag and drop this block in the design area, a selection dialog box appears with which you can define the input and output data type.</p> <p>ST: Function name is formed from <i>Source_type_to_Target_type</i>, e.g. <code>OUT := real_to_int(IN);</code></p>



Note

Conversion to TIME:

Integer values are interpreted as millisecond values.

Real values are interpreted as seconds values.

Example:

An integer value of 4,711 hence, yields 4,177 seconds.

A real value of 4711.0 yields 4711 seconds (that is, 1 h, 18 min and 31 sec).

20 Error Codes

20.1 DAT_FILE_WRITE Error Codes

Error Code	Meaning
16#FFFFFFF1	No PP_COMMAND specified!
16#FFFFFFF2	Failed to execute PP_COMMAND!
16#FFFFFFF3	Failed to start PP_COMMAND!
16#FFFFFFF4	Failed to close file xxxx.dat
16#FFFFFFF5	Sample Time is set to 0.0. Please set a valid sample time!
16#FFFFFFF6	DatfileWrite Configuration exceeds allowed signals in Dongle
16#FFFFFFF7	Failed to write data into file. Please check disk space
16#FFFFFFF8	Failed to create file. Please check file name and disk space!
16#FFFFFFF9	Could not find Data handlers for module x
16#FFFFFFFA	Error collecting Data handlers for module x
16#FFFFFFFB	Could not find Data handlers
16#FFFFFFFC	Could not find Module configuration data for module x
16#FFFFFFFD	Error reading Module configuration data for module x
16#FFFFFFFE	No Module Configuration defined!
16#FFFFFFF	Could not find Configuration data

20.2 TCPIP_SENDRECV Error Codes

Error Code	Meaning	Solution suggested
HEX: 16#0000271d DEZ: 10013	Permission denied - Access to socket forbidden by access permissions.	Please login with a username that has administrator privileges.
16#00002740 10048	Address already in use - Only one usage of each socket address is permitted.	The address / port is already used.
16#00002741 10049	Cannot assign requested address - The requested address is not valid in its context.	
16#0000273f 10047	Address family not supported by protocol family - An address incompatible with the requested protocol was used.	
16#00002735 10037	Operation already in progress - An operation was attempted on a non-blocking socket that already had an operation in progress.	
16#00002745 10053	Software caused connection abort - An established connection was aborted by host machine.	
16#0000274d 10061	Connection refused - No connection could be made because the target machine actively refused it.	

Error Code	Meaning	Solution suggested
16#00002746 10054	Connection reset by peer - An existing connection was forcibly closed by the remote host.	
16#00002737 10039	Destination address required - A required address was omitted from an operation on a socket.	
16#0000271e 10014	Bad address - The system detected an invalid pointer address in attempting to use a pointer argument of a call.	
16#00002750 10064	Host is down - A socket operation failed because the destination host was down.	
16#00002751 10065	No route to host - A socket operation was attempted to an unreachable host.	
16#00002734 10036	Operation now in progress - A blocking operation is currently executing.	
16#00002714 10004	Interrupted function call - A blocking operation was interrupted by a call to WSACancelBlockingCall.	
16#00002726 10022	Invalid argument - Some invalid argument was supplied.	
16#00002748 10056	Socket is already connected - A connect request was made on an already connected socket.	
16#00002728 10024	Too many open files - Too many open sockets.	
16#00002738 10040	Message too long - A message sent to socket was larger than the internal message buffer or the buffer used to receive was smaller than the datagram itself.	Reduce the length of the bytes to be transmitted.
16#00002742 10050	Network is down - A socket operation encountered a dead network.	
16#00002744 10052	Network dropped connection on reset - The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress.	
16#00002743 10051	Network is unreachable - A socket operation was attempted to an unreachable network.	
16#00002747 10055	No buffer space available - An operation could not be performed because the system lacked sufficient buffer space or because a queue was full.	
16#0000273° 10042	Bad protocol option - An unknown, invalid or unsupported option or level was specified in a getsockopt or setsockopt call.	
16#00002749 10057	Socket is not connected - A request to send or receive data was disallowed because the socket is not connected.	
16#00002736 10038	Socket operation on non-socket - An operation was attempted on something that is not a socket.	
16#0000273d 10045	Operation not supported - The attempted operation is not supported for the type of object referenced.	
16#0000273e 10046	Protocol family not supported - The protocol family has not been configured into the system or no implementation for it exists.	

Error Code	Meaning	Solution suggested
16#00002753 10067	Too many processes - A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously.	
16#0000273b 10043	Protocol not supported - The requested protocol has not been configured into the system, or no implementation for it exists.	
16#00002739 10041	Protocol wrong type for socket - A protocol was specified in the socket function call that does not support the semantics of the socket type requested.	
16#0000274a 10058	Cannot send after socket shutdown - A request to send or receive data was disallowed because the socket had already been shut down.	
16#0000273c 10044	Socket type not supported - The support for the specified socket type does not exist in this address family.	
16#0000274c 10060	Connection timed out - A connection attempt failed or established connection failed because connected host has failed to respond.	
16#0000277d 10109	Class type not found - The specified class was not found.	
16#0000277a 10106	Unable to initialize a service provider - Either a service provider's DLL could not be loaded or the provider's WSPStartup/NSPStartup function failed.	
16#00002af9 11001	Host not found - No such host is known.	
16#0000276d 10093	Successful WSASStartup not yet performed - Either the application hasn't called WSASStartup or WSASStartup failed.	
16#00002afc 11004	Valid name, no data record of requested type - The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for.	
16#00002afb 11003	This is a non-recoverable error - This indicates some sort of non-recoverable error occurred during a database lookup.	
16#0000277b 10107	System call failure - Returned when a system call that should never fail does.	
16#0000276b 10091	Network subsystem is unavailable - The underlying system to provide network services is currently unavailable.	
16#00002afa 11002	Non-authoritative host not found - Temporary error during hostname resolution, the local server did not receive a response from an authoritative server.	
16#0000276a 10092	WINSOCK.DLL version out of range - The current Windows Sockets implementation does not support the Windows Sockets specification version requested by the application.	
16#00002775 10101	Graceful shutdown in progress - The remote party has initiated a graceful shutdown sequence.	
16#00002778 10104	Invalid procedure table from service provider - A service provider returned a bogus proc table to WS2_32.DLL.	

Error Code	Meaning	Solution suggested
16#00002779 10105	Invalid service provider version number - A service provider returned a version number other than 2.0.	
16#00002733 10035	Resource temporarily unavailable - Operation should be retried later.	
16#00000006 6	Specified event object handle is invalid - An application attempts to use an event object, but the specified handle is not valid.	
16#00000008 8	Insufficient memory available - The Win32 Socket function is indicating a lack of required memory resources.	
16#00000057 87	One or more parameters are invalid - The Win32 Socket function is indicating a problem with one or more parameters.	
16#000003e3 995	Overlapped operation aborted - An overlapped operation was canceled due to the closure of the socket.	
16#000003e4 996	Overlapped I/O event object not in signaled state - The application has tried to determine the status of an overlapped operation which is not yet completed.	
16#000003e5 997	Overlapped operations will complete later - The application has initiated an overlapped operation which cannot be completed immediately.	

21 Characteristics of TCP/IP

21.1 Number of TCP/IP connections possible



Note

The number of TCP/IP connections possible in ibaLogic depends on the system settings "TcpNumConnections".

An excerpt of Microsoft for this:

"TcpNumConnections"

Registry:

HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

Keyname: TcpNumConnections

Data type	Range	Default value
REG_DWORD	0x0 – 0xFFFFFE	0

Description

Determines the maximum number of connections that TCP can have open simultaneously. If the value of this entry is 0, you cannot establish any connections.

Note Image Note

Windows does not add this entry to the registry. You can add it by editing the registry or by using a program that edits the registry."

LINKS

TcpNumConnections (<http://technet.microsoft.com/en-us/library/cc938216.aspx>)

TCP/IP - Maximum number (<http://www.windowspage.de/tipps/021202.html>) of simultaneously open connections

21.2 Delayed Acknowledge Problem

Problem

Measurements made by automation equipment using TCP/IP do not work with cycle times < 200 ms.

Error pattern: Sequence error, incomplete telegrams and different lengths received.

Cause

There are different variants of handling 'Acknowledge' in the TCP/IP protocol:

1. The standard WinSocket works in accordance with RFC1122 using the "delayed acknowledge" mechanism. This states that the acknowledge is delayed until other telegrams arrive in order to acknowledge them jointly. If no other telegrams arrive, the ACK telegram is sent latest after 200 ms (depending on the socket).

According to the TCP/IP standard this is possible since with data flow control using "Sliding Window" (Parameter Win = nnnn) the receiver specifies the number of bytes that it can receive without sending an acknowledgment.

2. Some controllers do not accept this response, but instead, wait for an acknowledgment after each data telegram. If this does not arrive within a specific time period (200 ms), it repeats the telegram and possibly also adds new data to be transmitted. This leads to an error in the receiver, since the older telegram was received correctly.

Remedy

The "delayed acknowledge" must be disabled in Windows with the help of the parameter entry in the Windows registry:

"TcpAckFrequency" REG_DWORD = 1;

The parameter is not present by default and has to be entered in this path:

"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{InterfaceGUID}"



Note

You must select the correct interface. You can see which one is the correct one, for example, from the IP addresses set currently.

See also the following MS

New registration entry (<http://support.microsoft.com/kb/328890>) for checking the TCP confirmation response in Windows XP and Windows Server 2003

Windows XP:

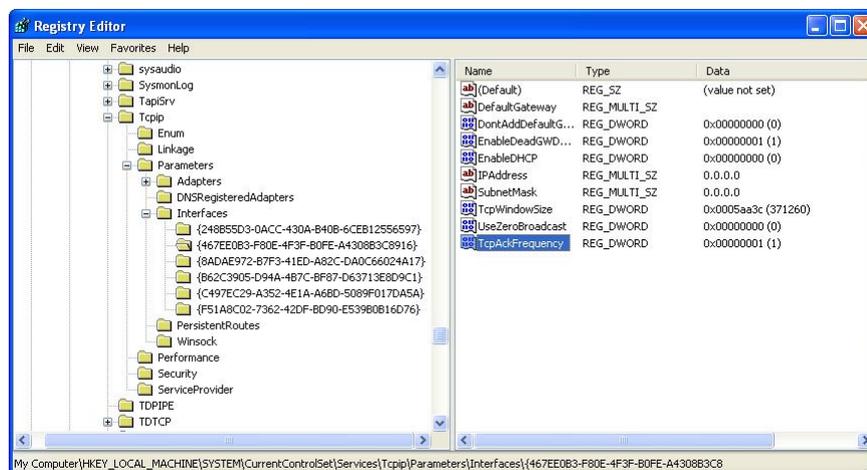


figure 148: Windows XP Registry

22 Key Combinations

22.1 Client

Key combination	Explanation
<Ctrl> + <P>	Print
<Ctrl> + <Z>	Undo
<Ctrl> + <Y>	Repeat
<Ctrl> + <C>	Copy
<Ctrl> + <V>	Paste
<Ctrl> + <A>	Select all
	Remove
<F5>	Start evaluation
<Shift> + <F5>	Stop evaluation
<Ctrl> + <Shift> + <F>	Add – new function block
<Ctrl> + <Shift> + <M>	Add – new macro block
<Ctrl> + <Shift> + <I>	Add – new intra-page connector
<Ctrl> + <Shift> + <T>	Add – new off-task connector
<Ctrl> + <Shift> + <C>	Add – new comment
<Ctrl> + <Shift> + <S>	Display off-task connectors
<F1>	Help

22.2 Mouse Functions in the Programming Field

Key combination	Explanation
Left mouse button click + <Shift>:	Mark multiple elements
Left mouse button click + <Ctrl>	Switch the marking
Scroll wheel + <Shift>:	Move visible section to the left / right.
Scroll wheel + <Ctrl>:	Zoom in / zoom out
Scroll wheel + <Alt>	Move visible section up / down.
<Ctrl> + connector input or output	Create IPC
Drag & Drop + <Alt>	You can drag signals by keeping the <Alt> button pressed from a connector and drop them into the ibaPDA Express window.

22.3 ibaPDA Express

Key combination	Explanation
<F6> (Switch)	Starts continuous display with the current time point. Active, when "Stop scrolling" is pressed.
<F6> (Switch)	Stop the continuous display. After pressing this, a ruler appears in graphs that can be moved with the mouse and with which the curves can be measured. The signal values are displayed in the legend. You can move the X-axis using the mouse. In this manner, you can browse values from the past. Active, when the display is on.
<F5>	"Auto scale"
<F3>	Active only when the display has been zoomed. Return to the previous zoom factor (reduce).
<F4>	Active only when the display has been zoomed. Return to the initial (automatic) display.
<F10>	Exit from the full screen mode.

23 Character tables

ibalogic uses simplified Hex coding.

Thus, the first 256 Unicode characters (U+0000 to U+00FF) can be displayed.

Standard ASCII character table (\$00 - \$7F)

\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07
\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F
\$10	\$11	\$12	\$13	\$14	\$15	\$16	\$17
\$18	\$19	\$1A	\$1B	\$1C	\$1D	\$1E	\$1F
\$20	\$21	\$22	\$23	\$24	\$25	\$26	\$27
	!	"	#	\$	%	&	'
\$28	\$29	\$2A	\$2B	\$2C	\$2D	\$2E	\$2F
()	*	+	,	-	.	/
\$30	\$31	\$32	\$33	\$34	\$35	\$36	\$37
0	1	2	3	4	5	6	7
\$38	\$39	\$3A	\$3B	\$3C	\$3D	\$3E	\$3F
8	9	:	;	<	=	>	?
\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47
@	A	B	C	D	E	F	G
\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F
H	I	J	K	L	M	N	O
\$50	\$51	\$52	\$53	\$54	\$55	\$56	\$57
P	Q	R	S	T	U	V	W
\$58	\$59	\$5A	\$5B	\$5C	\$5D	\$5E	\$5F
X	Y	Z	[\]	^	_
\$60	\$61	\$62	\$63	\$64	\$65	\$66	\$67
`	a	b	c	d	e	f	g
\$68	\$69	\$6A	\$6B	\$6C	\$6D	\$6E	\$6F
h	i	j	k	l	m	n	o
\$70	\$71	\$72	\$73	\$74	\$75	\$76	\$77
p	q	r	s	t	u	v	w
\$78	\$79	\$7A	\$7B	\$7C	\$7D	\$7E	\$7F
x	y	z	{		}	~	

Extended character table (\$80 - \$FF)

\$80	\$81	\$82	\$83	\$84	\$85	\$86	\$87
€		,	f	„	…	†	‡
\$88	\$89	\$8A	\$8B	\$8C	\$8D	\$8E	\$8F
ˆ	‰	š	˘	œ		ž	
\$90	\$91	\$92	\$93	\$94	\$95	\$96	\$97
	˙	˚	“	”	•	–	—
\$98	\$99	\$9A	\$9B	\$9C	\$9D	\$9E	\$9F
˜	™	š	˚	œ		ž	ÿ
\$A0	\$A1	\$A2	\$A3	\$A4	\$A5	\$A6	\$A7
	ı	ç	£	¤	¥	¦	§
\$A8	\$A9	\$AA	\$AB	\$AC	\$AD	\$AE	\$AF
¨	©	ª	«	¬		®	-
\$B0	\$B1	\$B2	\$B3	\$B4	\$B5	\$B6	\$B7
°	±	²	³	´	µ	¶	·
\$B8	\$B9	\$BA	\$BB	\$BC	\$BD	\$BE	\$BF
¸	¹	º	»	¼	½	¾	¿
\$C0	\$C1	\$C2	\$C3	\$C4	\$C5	\$C6	\$C7
À	Á	Â	Ã	Ä	Å	Æ	Ç
\$C8	\$C9	\$CA	\$CB	\$CC	\$CD	\$CE	\$CF
È	É	Ê	Ë	Ì	Í	Î	Ï
\$D0	\$D1	\$D2	\$D3	\$D4	\$D5	\$D6	\$D7
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×
\$D8	\$D9	\$DA	\$DB	\$DC	\$DD	\$DE	\$DF
Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
\$E0	\$E1	\$E2	\$E3	\$E4	\$E5	\$E6	\$E7
à	á	â	ã	ä	å	æ	ç
\$E8	\$E9	\$EA	\$EB	\$EC	\$ED	\$EE	\$EF
è	é	ê	ë	ì	í	î	ï
\$F0	\$F1	\$F2	\$F3	\$F4	\$F5	\$F6	\$F7
ð	ñ	ò	ó	ô	õ	ö	÷
\$F8	\$F9	\$FA	\$FB	\$FC	\$FD	\$FE	\$FF
ø	ù	ú	û	ü	ý	þ	ÿ

24 Index of Abbreviations

Abbreviation	German	English
ASCII	Zeichenkodierung	American Standard Code for Information Interchange
AWL	Anweisungsliste	Statement List
CE	Übereinstimmung mit EU-Richtlinien	FR.: Conformité Européenne
CFC	Funktionsblockdiagramm	Continuous Function Chart
CPU	Prozessor	Central Processing Unit
CR	Wagenrücklauf	Carriage Return
CSV		Comma Separated Values or Character Separated Values
DA	Datenzugriff	Data Access
DCOM		Distributed Component Object Model
DFW		DAT_FILE_WRITE
DIN	Deutsches Institut für Normung	
DLL	Dynamische Verbindungsbibliothek	Dynamic Link Library
E/A	Eingang/Ausgang	INPUT/OUTPUT
FB	Funktionsbaustein	Function Block
FBD	Funktionsblockdiagramm	Function Block Diagram
FFT		Fast Fourier Transformation
FOB	Lichtwellenleiterkarte	Fiber optical board
FUP	Funktionsplan	Function Chart
GDM		Global Data Memory
GMT		Greenwich Mean Time
GSD	Gerätstammdaten-Datei (Profibus)	Generic Station Description
HMI	Bedien-Beobachten-System	Human Machine Interface
HW	Hardware	Hardware
I/O	Eingang/Ausgang	Input/Output
IEC	ein Normungsgremium für Elektrotechnik	International Electrotechnical Commission
IL	Anweisungsliste	Instruction List
IP	Internet-Protokoll	Internet Protocol
IPC	Intra-Page-Konnektor	Intra-page connector
KOP	Kontaktplan	
LAD		Ladder Diagram
LF	Zeilenvorschub	Line Feed
FOC	Lichtwellenleiter	Fiber optical conductor
MB	Makroblock	Macro Block
MDAC		Microsoft Data Access Components
MS SQL		Microsoft Structured Query Language
NL	Zeilenvorschub	Newline
OLE	Objekt-Verknüpfung und -Einbettung	Object linking and embedding
OPC	Datenaustauschprotokoll (Schnittstelle)	OLE for Process Control
OTC	Off-Task-Konnektor	Off Task Connector
PAC	Programmierbare Automatisierungseinheit	Programmable automation controller
PADU	Parallel-Analog-Digital-Umsetzer (Varianten: PADU-S, PADU-S-IT)	Parallel analog digital unit

Abbreviation	German	English
PC	Einzelplatzrechner	Personal Computer
PCI	PC-Bussystem	Peripheral Component Interconnect
PDA	Prozess-Daten-Aufzeichnung	Process data acquisition
PLC	Siehe SPS	Programmable Logic Controller
PMAC	Programmierbare Mess- und Automatisierungseinheit	Programmable Measurement and Automation Controller
RAM	Arbeitsspeicher	Random Access Memory
RFM		Reflective Memory
SD	SIMADYN D	SIMADYN D
SPS	Speicherprogrammierbare Steuerung	See PLC
SST	Profibuskarte	
ST	Strukturierter Text	Structured Text
STL	Anweisungsliste	Statement List
SW	Software	Software
Tab	Tabulator	Tabulator
TCP/IP	Netzwerk-Protokoll	Transmission Control Protocol/Internet Protocol
TDC	Automatisierungssystem (Siemens)	
UCODE	Bytecode	
USB		Universal Serial Bus
UTC	koordinierte Weltzeit	Universal Time Coordinated
WDM		Windows Driver Model
XML		Extensible Markup Language
XP	Windows XP	Windows XP

25 Classified Index

A		Data types	139
Access synchronization	68	derived	278
Application	32	generic	279
Comments	34	Standard	278
Data types	34	Database	
Function blocks	33	Configure connection	42
Graphics Programming	34	Configure the interface	44
ibaPDA Express	35	reset	216
Program Elements	33	restore	214
Structure	32	save	210
Task / Program Properties	32	manually	210
Autostart	47, 50	Database Management	210
B		Debugging	217, 237
Blocks	90	Compilation errors	238
export	93	Errors in user-defined function blocks	237
import	94	Evaluation Order	237
manage	93	Incorrect signal trends	237
remove	95	Program errors	237
use	91	Define Group	81
C		Definition	58
Circuit		Definition name	124
Connect	258	Definitions	59
Switch online	256	Disconnect	169
Test	257	DLL	
Comments	164	Descriptions	137
Compiler	136	establish	136
Configuring the client port	41	Integration	137
Connect	169	Notes	137
Connection lines		Prerequisites	137
establish	154	Source files	137
modify	155	E	
Converter	162	Error Codes	
D		DAT_FILE_WRITE	314
DAT_FILE_WRITE		TCPIP_SENDRECV	314
Error Codes	314	Evaluation sequence	60
Mode		Rules	60
Buffered	271	Events Window	68
Unbuffered	267	All Events	69
Sample project	267	Console View	69
Data base scripts		Local Events	69
manage	46	Server events	69
Data Type		F	
delete	142	FOB cards	
export	143	Buffered Mode	192
group	139	FOB-4io-S card	186
during the creation of a function block.	141	iba-FOB-io-S card	182
In the global library	141	Function block	123
Under the project	141	Analytical Functions	282
import	144	Arithmetical Functions	
manage	142	Bistable	288
names	142	Bit String	289
use	144	Bitwise_Boolean	290
during the creation of a function block.	144	Character String	291
User	144	Communication	292
Data Type:	145, 146, 148, 149	Comparison	293
		Counter	295
		Edge Detection	296
		Miscellaneous	286
		Register	297
		Selection	298

Signal Processing	300	Events Window	68
Specials	301	All Events	69
Timer	304	Console View	69
Trigonometric	285	Local Events	69
complex	96	Server events	69
Data Types	280	Hierarchy	60
Function diagram display	280	Instances	58
General Settings	124	Menu bar	55
Place	252	Navigation Area	56
Set parameters	255	Program Designer	62
Standard	96, 280	Programming Environment	54
Type Conversion	306	start	54, 250
Scaling Converter	311	Toolbar	55
user-specific	123	User Interface	55
G		Workspace	70
Graphical Connections		Workspace Explorer	57
Connection lines	154	ibaLogic Server	37
Connector types	154	Configuring the client port	41
H		Configuring the Database Connections	42
Hardware configuration		Configuring the Database Interface	44
Driver restart	180	Data base scripts	46
Interrupt source	180	Function overview	37
Measurement	180	General Settings	49
Soft PLC	180	Language	52
Time base	180	PMAC settings	50
Turbo mode	180	Select SQL server	45
Watchdog	180	Setting	41
Hierarchy	60	start	38, 250
I		Status bar	53
I/O configurator	175	User interface	40
Assign signals	175	ibaLogic Server:	47
Hardware configuration	175	ibaLogic Software	
Input / output resources	175	Measurement &	
ibaLogic		Condition monitoring	25
Areas of application	25	ibaPDA	
Automation	25	Toolbar	228
PLC co-processor	25	ibaPDA Express	217
Signal management	25	Color signal	220
Simulator	25	Extended Functionality	228
Components	27	Measured value storage	35
ibaLogic Client	27	Move Scales	222
ibaLogic Server	27	Move Signal	219
OPC Server	27	Remove Graphs	220
Runtime system (PMAC)	27	Remove Signal	220
Connectivity	36	Sample exercise	260
Identification	13	Scale Axes	221
License Activation	16	Select Signals	218
Profibus master	196	Trend Graph	218
Card settings	196	Trend Graph Properties	224
Configuration	196	Zoom Function	223
Profibus slave	195	Input / Output variables	
Card settings	195	create	153
Proper Use	13	Input resources	192
Release notes	13	Inputs and Outputs	
Software	24	configure	80
ibaLogic Client	54	Installation	17
<Read write> / <Read only> button	68	Choose components	17
Definitions	59	Choose Installation Location	17
Evaluation sequence	60	Define the start menu folder	17
		License agreement	17
		Select SQL server	17
		Software required	17
		System Requirements	17

Instance	58	Output resources	192
Instance name	124		
Instances	58	P	
Integrated measurement using ibaPDA Express		PADU-S-IT	178, 204
	35	Card settings	204
Intra-Page Connectors	155	Settings	204
establish	155	PCI Interfaces	190
Modify name	156	Card settings	190
track	156	Performance Limits	240
J		PIDT1_CONTROL	107
Joiner	163	Example	110
K		Inputs	109
Key Combinations		Outputs	109
Client	320	Signal trends	110
ibaPDA Express	321	Platform	
Programming field	320	configure	172
L		PADU-S-IT	171, 178
Language	52	select	174
Link settings	190	WinXP	171
Local peripherals	204	Platforms	171
M		PMAC	165
Macro block	133	Settings	50
combine	134	PMAC memory	
create	133	delete	168
expand	135	save	167
open	134	Processing modes	31
Menu bar	55	Program	
Mode		create	76
Offline	165	names	77
Online	165	open	77
Modify signal assignment	188	remove	78
Multi-client operation	27	Program analysis	259, 260
N		Program clarity	
Naming conventions	277	Sample exercise	261
Navigation Area	56	Program Creation	90
O		Program Designer	62
Off-Task Connectors	157	Arrangement of the programming windows	63
Display	161	Arrangement of the Tabs	63
establish	157	Evaluation context	63
List	161	Navigate	65
rename	159	Program overview	65
track	160	Toolbar	63
OPC		Program Elements	151
Communication	207	Graphical Connections	154
Server	207	Programming Environment	54
Set variable parameters	209	Programming rules	242
Operating modes	31	project	
Buffered mode	31	create	251
Buffered Mode	271	Project	73
Measurement	31, 180, 230	create	73
Soft PLC	31, 180, 230	load	75
Turbo mode	180	remove	75
Unbuffered mode	267	Set Project as Active	74
OTC		Project Properties	75
establish	258	R	
		RAMP	116
		Example	118
		Inputs	117
		Outputs	117
		Read write / Read only	68
		Reflective Memory	201

Card settings	201	create	76
Configuration	201	open	77
Parametrization	201	remove	78
Resources	176	TCPIP_SENDRECV	105
Global System Variables	179	Error Codes	314
Hardware	178	Time behavior	217, 230
Hardware configuration	180	Toolbar	55
Card settings	181		
General Settings	180	U	
Software	179	User interface	
Update Hardware	176	ibaLogic Server	40
Runtime system	165	User Interface	
Autostart	167, 168	ibaLogic Client	54
Connect	169	User-defined block	
Disconnect	169	create	92
start	165	In the global library	92
stop	166	Under the project	92
		User-defined Data Types	144
S		V	
Sample exercise		Views	
Getting started	249	Definitions	59
ibaPDExpress	260	Evaluation sequence	60
Program clarity	261	Hierarchy	60
Structured Text	262	Instances	58
Sample project		W	
DAT_FILE_WRITE	267	Workspace	
Scientific notation	224	close	71
Set operating mode	31	create	70
Set processing modes	31	delete	72
Signal		open	71
assign	182	Workspace Explorer	
create	81	IEC View	57
edit	84	Prog view	57
export	86	Task view	57
group	82	Workspaces	70
import	86		
remove	84, 89		
use	88		
Signal name			
define	188		
modify	182		
SIMADYN D / SIMATIC TDC Connection	199		
Card settings	199		
Link settings	199		
Software Installation	14		
Splitter	163		
Structured Text			
Editor	127		
IntelliSense	128		
Sample exercise	262		
Syntax description	128		
Syntax description			
Constants	128		
Operators	128		
Statements	128		
Strings	128		
System Requirements	14		
Hardware	14		
Software	15		
T			
Task	76		

26 Support and contact

Support

Phone: +49 911 97282-14

Fax: +49 911 97282-33

Email: support@iba-ag.com



Note

If you require support, indicate the serial number (iba-S/N) of the product.

Contact

Headquarters

iba AG
Koenigswarterstr. 44
90762 Fuerth
Germany

Phone: +49 911 97282-0

Fax: +49 911 97282-33

Email: iba@iba-ag.com

Contact: Mr Harald Opel

Regional and Worldwide

For contact data of your regional iba office or representative please refer to our web site

www.iba-ag.com.