ibaLogic

Programming software for signal management, simulation and soft-plc

Manual

Version 4.3 en / ibaLogic 3.90c

Measurement and Automation Systems

iba

ibaLogic Manual

Issued by

iba AG
Koenigswarterstr. 44
90762 Fuerth, Germany

Tel.: + 49 (0)911 9 72 82-0
Sales -27
Support -14
R&D -13
FAX -33
Email: iba@iba-ag.com
Web: www.iba-ag.com

2th revised edition, ibaLogic Manual V 4.2 en / ibaLogic 3.88b

We have checked that the contents of this manual match the hardware and software described here. However, deviations cannot be fully ruled out, so that we cannot assume any warranty should any deviations actually exist. This manual is regularly updated. Necessary revisions are included in future editions, or can be downloaded from the Internet.

The latest version is always available for downloading at:
http://www.iba-ag.com.

We would welcome any suggestions for improvements which you may have.

Version	Date	Revision	Chapter / pages	Author	Version ibaLogic
V 4.3	Feb 12 2009	ibaLogic-V3-Runtime	1, 2, 4 , 5	ko	3.90c

Contents

Foreword

This compact manual provides the information for handling the graphical programming software **ibaLogic**.

The operation of the software is explained for many cases by using typical examples. In particular cases especially in conjunction with process in- and output components please refer also to the related hardware documentation.

You can find the latest issue of this manual always on our website http://www.iba-ag.com in the download area.

This manual contains seven chapters explaining the use of ibalogic and its features.

Chapter 1 In the first chapter you'll find an introduction with information about the most important features of ibaLogic and the standard IEC1131-3.

Chapter 2 This chapter describes the user interface with all menus and dialog windows. The most important settings of the program and the system are described here.

Chapter 3 In chapter 3 you'll find practical advise for working with ibaLogic. Stages of operation from program design over usage of function blocks, creation of macros, testing and debugging up to printing are described in detail.

Chapter 4 All standard function blocks and functions which are available in ibaLogic are listed and explained in this chapter.

Chapter 5 In this chapter you'll find the description of the in- and output resources and OPC communication.

Chapter 6 System requirements and software installation as well as some special features when using former ISA-boards are the subject of this chapter.

Chapter 7 In the last chapter you'll find additional information for special topics, such as program listings, dedicated application examples etc.

Finally, this manual also contains a glossary which serves as a quick-finding reference to special terms and abbreviations, a list of references and an index that can help to quickly find the information you need.

This manual uses several symbols which essentially have the following meanings:

Important hint or warning in order to avoid hazard against material or life.

A useful tip or clue to make your work easier.

This draws your attention to special features, such as exceptions to rules, etc.

A reference to additional documentation or more in-depth literature.

Software or file name

reference to associated software or sample applications on the CD-ROM.

iba training courses

Hint for training courses by iba concerning related products or subjects

The following notation refers to menu functions in ibaLogic:

↪*File* ↪*System settings*

Wenn using the trm "mouseclick" we always refer to the left mousekey. In case the right mousekey should be used it's pointed out.

The software ibaLogic works only with operating systems MS Windows® NT 4.0, MS Windows® XP or MS Windows® 2000.

MS Windows® NT, 2000 and XP are registered trademarks of the Microsoft Corporation.

1 Welcome to ibaLogic

1.1 Introduction

ibaLogic combines the convenience of a comfortable signal manager and the performance of a powerful soft-plc. Because ibaLogic is often used for high speed measuring and control applications, very short scan cycles (≥ 1 ms) and a time-deterministic behaviour are essential system properties.

Beside an easy handling the great advantage of ibaLogic is the exclusive use of international standards in terms of operating system, communications and programming language which guarantees the openness, portability and reusability of application programs created with ibaLogic.

Standard-PCs with Windows® as operating system are the hardware platform for ibaLogic. As a consequence ibaLogic benefits from all current and future developments in the PC industry, such as internet technology, remote access and, of course, the continuing increase of processor performance.

Using a diagrammatical programming language with function block diagrams makes it very easy for the user to build an application with ibaLogic. Of course, ibaLogic complies with the requirements of the IEC 61131-3 standard for soft-plc. The reasons are not only the portability, the easy-to-learn effect or market strategy. Moreover, ibaLogic offers a wide range of solutions for program design and applications by using consequently the data formats and languages of IEC 1131-3, e.g. "Structured Text (ST)" as meta language or "STRING" as a convertible data format.

The flexible process interface and the open communication interface are two of the major advantages of ibaLogic. The connection to sensors and actors is either done by international standardized field bus systems (e.g. Profibus), by using the ibaNet750 I/O-system with components from WAGO/Beckhoff or by using fast PADU units (Parallel Analog Digital Converter) for control or regulation. The open communication between ibaLogic and HMI-systems or other higher level computers works with a standardized OPC interface and TCP/IP or "Named Pipes".

ibaLogic-V3-Runtime is the economy-priced version of ibaLogic-V3. ibaLogic-V3-Runtime is used to execute only a runtime without a possibility to edit the program with an editor.

The general applications for ibaLogic are:

Fast signal (pre-)processing and signal distribution

- Signal management and signal preprocessing for ibaQDA, ibaPLR or ibaVision
- Signal preparation and complex trigger-generation for ibaPDA, ibaQDR, ibaPLR or ibaQDA
- Fast signal switching and management between ibaLogic and other applications (e.g. ibaQDA or Visual C++ or Visual Basic programs written by the user himself)

1

Soft-plc in compliance with IEC 61131-3

- PC-based automation system for Windows® with ibaLogic as high-class soft-plc.
- Due to its easy handling and intuitive operation and due to its versatile interfaces and integrated fast online monitoring features ibaLogic meets perfectly the requirements of revamping existing control applications. If these existing control applications were written in Structured Text they could be even reused by ibaLogic.

Signal processing

- Condition monitoring system for machines
- Vibration analysis for machines, with sampling rates of up to 25 kHz / channel
- Monitoring for bearings and alarm message generation
- New ways of quality data recording and monitoring
- Storing signals in iba's **.dat* file format and retrieving (playback) recorded data

Simulation

- Simulation of rolling mill stands, e.g. for training purposes
- Simulation of entire plants, e.g. for testing control and regulation applications in other automation devices

IEC1131-3 Software-Development-Package

- Platform-independent programming language, based on IEC 61131-3 standards (ST)

1.2 System properties of ibaLogic in brief

- ❑ Shortest program cycletime is 1 ms and higher
- ❑ Time-deterministic behaviour with Windows®
- ❑ Userfriendly by Windows-like look-and-feel, easy to learn and to handle; graphic programming with autorouting support;
- ❑ Short turn around time for operation inputs or program modifications. These actions are executed immediately without compilation. If these inputs or modifications are performed in the online-layer they will directly affect the process (!) ("...like wiring a former control cabinet under voltage").
- ❑ HOT SWAP switching, i.e. it's possible to modify a functionblock diagram while the current program version is still controlling the process. When the modification is finished a smooth switch-over will activate the new program version. This feature is a big advantage particularly for continuous processes, e.g. in the paper industry or processing lines.
- ❑ Programming language, data formats and the functionblock library are in compliance with the international standard IEC1131-3.
- ❑ The following data types and formats are supported:
  Boolean, Integer (16 bit, 32 bit, unsigned 32 bit), double word, float (32 bit, 64 bit), string, time and array (4-dimensional of the previous mentioned datatypes, except string, homogeneous)
- ❑ An extensive function block (FB) library with many standard and special functions. A further extension of the library by the user himself is possible.

- Two methods of creating new function blocks interactively:
  - Simply, without deeper programming knowledge, by using mathematic formulas
  - Extension of the first method by using the "Structured Text (ST)" meta language. Thus, it's possible to use “if-then-else”-queries or “for-next” loops.
- Program structuring by means of "macro blocks" (MB), made of one level or interlaced; simple creation of MBs by marking and combining several function blocks.
- An open DLL-interface in order to integrate special functions or technological know-how, e.g. by means of “C” or “C++” programs.
- Full support of a hierarchical program design by using the means creation and integration of macros.
- Support of “multitasking” and task-to-task-communication.
- A fully integrated product, i.e. all required tools and compilers (ST, C++, Assembler) are integrated in ibaLogic; easy installation and handling.
- Process I/O link for the following systems:
  - Input (typical: 1ms) of analog- / binary inputs with fibre optical link between FOB I/O or FOB 4i PCI boards (unidirectional) and PADU8/16/32.
  - Input of fast analog- / binary inputs (up to 25 kHz / channel) with FOB I/O-PCI or FOB 4i-PCI + FOB 4o, running in "FOB-M mode" (unidirectional), linked to Padu ICP / Padu M.
  - Input / output (typical: 1ms) of analog / binary inputs and analog / binary outputs with fibre optical link between FOB-IO (bidirectional) and PADU8/16/32 and Padu8-O, SLM.....
  - Input / output of analog / binary inputs and analog / binary outputs with fibre optical link between FOB-IO (bidirectional) and ibanet750-head module with connection to WAGO- terminals (I/O delay time is module-specific, see data sheet of I/O-modules), image copy of WAGO-head: typ. 1 ms.
  - Diverse interfaces to common fieldbus systems and backplanes, such as (Profibus-DP, VME-Bus, MMC/S5 u.a.).
  - Diverse interfaces to plc and control systems of the major brands, such as ABB, ALSTOM, SIEMENS, SMS-Demag, KVÆRNER, PROSOFT, ALLEN-BRADLEY etc.
- Open communication interface
  - TCP/IP by “named pipes” (for in- and outputs) in connection to PCs and plc-systems and *.csv –files (comma separated value), e.g. for use in MS Excel or other programs
  - OPC interface to standard HMI systems
  - TCP/IP communication to distributed ibaPDA/ibaQDA and / or ibaLogic soft-plc applications
  - SIMATIC S7 by L2B card (Profibus DP slave module, uni- and bidirectional).
  - SIMATIC S5 or MMC216 by SM64-IO card, uni- and bidirectional.
  - Serial interface with 3964R protocol (e.g. Siemens process computer of R- and M-series).
  - Simatic TDC interface FOB TDC to GDM (bidirektional)
  - Simadyn-D interface card FOB SD to rack connection CS12/13/14
  - ALSTOM ALSPA C80 HPC (Logidyn D1, D2) by VME interface card SM128V
  - System connectors to CAN, Profibus Master, DeviceNet and ControlNet, coming soon

1

1.3 The plc programming languages according to IEC 1131-3

Before the introduction of IEC 61131-3 there was a variety of different programming languages for plcs which were not standardized and very often customized only for the devices of their manufacturer. Former program-linguistic means, such as Instruction List were not efficient enough and many solutions could have been easier with standard languages. Moreover, the periods of professional education of the maintenance staff were pretty long, particularly when getting familiar with existing plc applications. The lack of local memory ranges and of symbolic addressing lead to mistakes which were hard to find.

These deficits were part of the reasons for the definition of Part 3 in the IEC 1131 standard. In IEC 1131-3 the old languages had been standardized and finally supplemented by the new language "Structured Text" (ST). But the new standard describes not only the commands and syntax of a programming language. Furthermore, it declares the architecture and structure of a plc system from the software's point of view.

By means of the new languages, it is possible to describe a complete plc system, inclusive the hard and software assignment. The new lingual elements are *configuration*, *resource* and *task*. On the programming level there are the elements *program*, *function block* and *function*.

1.3.1. IEC 61131-3 software model

Statements of the norm with examples:

- An automation system consists of one or more *configuration*(s) which are able to communicate with each other. A configuration is, e.g., a plc rack with processor and I/O-cards or an ibaLogic-PC.
- A *configuration* consists of one or more *resources*. A resource is always assigned to one CPU only. One CPU can cover several resources. In ibaLogic there is always one resource per PC which is called "*Layout*". Layouts in ibaLogic are stored either in a **.lyt*-file or in a **.txt*-file (ST).
- One or more *tasks* can be assigned to one *resource*. A significant quality of a task is its cycle time. This period can be described explicitly. Several jobs with a mutual time base are combined in one task, e.g. all jobs to be activated in a 20 ms-period.

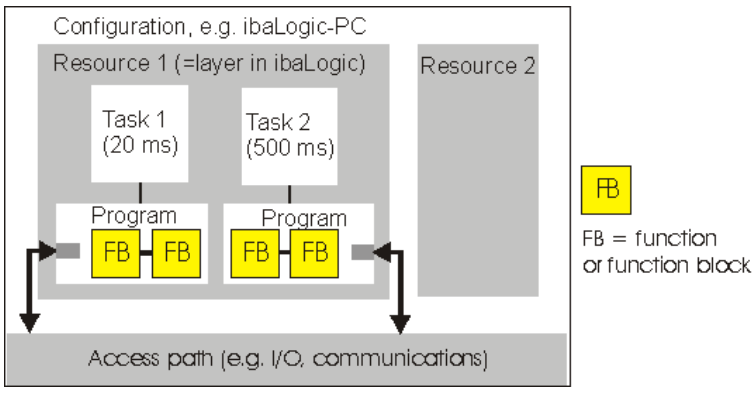


Fig. 1 IEC1131-3 software model

1.3.2. IEC 61131 program organization units (POU)

According to the IEC 1131 standard *functions*, *function blocks* and *programs* are program organization units (POU). One general restriction says that all POUs have to be non-recursiv, i.e. they should not call themselves in a program.

- *Functions* are subprograms which could have any input parameters but return only one result. Functions return always the same result for the same inputs (no memory effect).
- *Function blocks* can have many but clearly defined in- and output parameters and they can use internal variables, i.e. there is a memory effect. As an example for a function block a PID-regulator can be used multiple times in the same task or by different tasks with different sets of data.
- *Programs* contain the interconnection between functions and function blocks. A program can be written in any of the program languages which are defined in IEC 61131. The programs are explicitly assigned to a task with a certain period.

1.3.3. Supported datatypes

The following basic datatypes are supported by ibaLogic:

Typ	Range (min)	Range (max)	Remark
BOOL	0 (FALSE)	1 (TRUE)	
INT	-32_768	32_767	16-bit Integer (signed)
DINT	-2_147_483_648	2_147_483_647	32-bit Integer (signed)
UDINT	0	4_294_967_295	32-bit Integer (no sign)
DWORD	16#0000_0000	16#FFFF_FFFF	32-bit Word (no sign)
REAL	1.175_494_351 e-38	3.402_823_466 e+38	Floating point, single accuracy, 32 bit
LREAL	2.225_073_858_507_201_4 e-308	1.797_693_134_862_315_8 e+308	Floating point, double accuracy, 64 bit
TIME	-922_337_203_685_477_580.8 ms	922_337_203_685_477_580.7 ms	Time, internally depicted as 64-bit Integer (signed) with 0.1ms resolution per increment
STRING	0	1024 chs	String of characters with number of characters including terminal flag (NULL).
ARRAY	Structure, consisting of one of the above mentioned datatypes, except the String-type (which is an array by itself); maximum of four dimensions. Maximum number of elements: 1048576		

Table 1 Supported datatypes

2 Operation and setup

2.1 Getting started

2.1.1. ibaLogic-V3

If ibalogic is not installed on your PC yet, please refer to 6. There you will find a detailed description which guides you through the first steps of the installation.

ibaLogic is to be started simply by a double click on the file **ibaLogic*version*.exe** in the Windows® explorer. Depending on a customized installation there might be also an icon on your desktop screen or even an entry in the Windows® "Start menu" which could be used for program start.

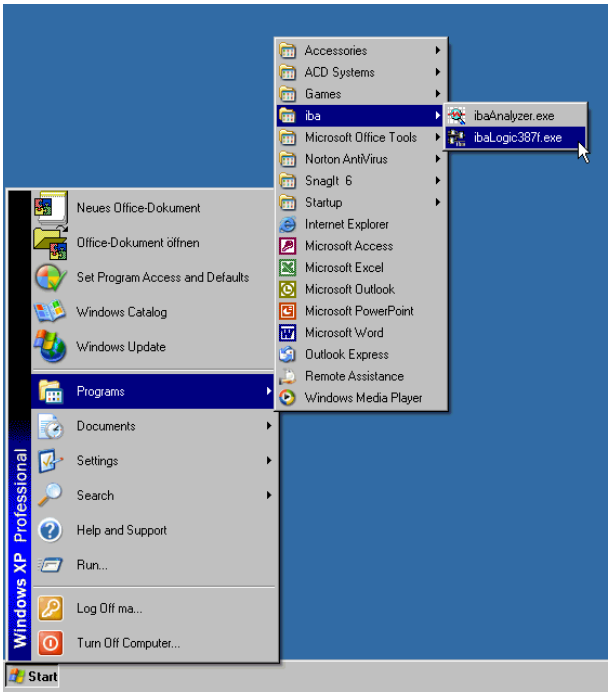


Fig. 2 Start of ibaLogic

If ibaLogic has been started without copy protection lock (dongle) a dialog window opens with some alternativs for starting ibaLogic even without dongle.

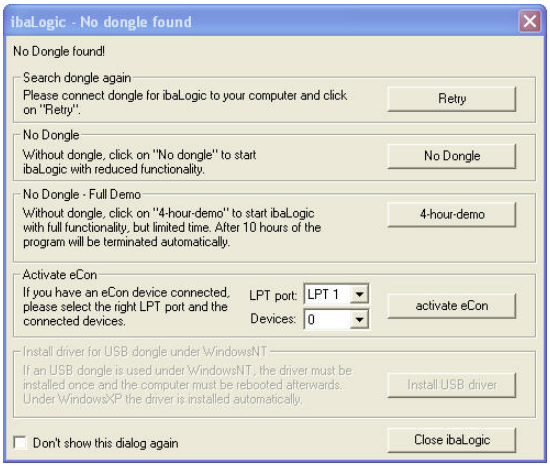


Fig. 3 Start of ibaLogic without dongle

In case you just forgot to attach the dongle please plug it now on the serial or USB interface and click on "Repeat search".

If no online mode and no playback function is requiered you may start ibaLogic without dongle.

ibaLogic would be started with full functionality, but limited time. After 4 hours of operation the program will be terminated automatically.

If ibaLogic should be used for operation together with an eCon no dongle is requiered either.

If you are working under Windows NT and want to use an USB-dongle but have not installed the USB drivers yet, you may do it now. After the installation of the USB-support just start ibaLogic again. When working under Windows XP or 2000 this option is disabled.

After the start the ibaLogic standard screen appears, with the major areas:

- ❑ Menu bar
- ❑ Tool bar
- ❑ Resource area with resource selection tabs
- ❑ Task area with task selection tabs

  Each task has an input and an output signal margin and a program area.

2.1.2. ibaLogic-V3-Runtime

ibaLogic-V3-Runtime is the economy-priced version of ibalogic-V3. ibaLogic-V3-Runtime is used to execute only a runtime without a possibility to edit the program with an editor. The runtime must be created with ibaLogic-V3 and copied on the process computer.

The file „autostart_runtime.lyt“ must be created with an ibaLogic-V3-System. ibaLogic-V3-Runtime and ibaLogic-V3 must to be of the same version.

ibaLogic-V3-Runtime has to be installed on the process computer, so the runtime can operate.

The installation procedure is the same as the installation of ibaLogic-V3.

If you create a runtime file, you have always to use “autostart_runtime.lyt” as filename. Copy the file “autostart_runtime.lyt” into the directory “…\schematics\” on the process computer.

The runtime file “autostart_runtime.lyt” is started automatically at the start of ibaLogic-V3-Runtime. If ibaLogic-V3-Runtime doesn’t find the file you will get an error message.

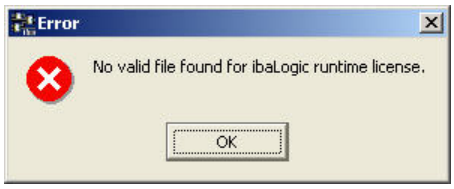


Fig. 4 Error message ibaLogic-V3-Runtime

ibaLogic-V3-Runtime is to be started simply by a double click at the icon on the desktop or a double click on the file ibaLogic*versionxy*.exe in the Windows® explorer.

The process and the state of the runtime are displayed at the state window.

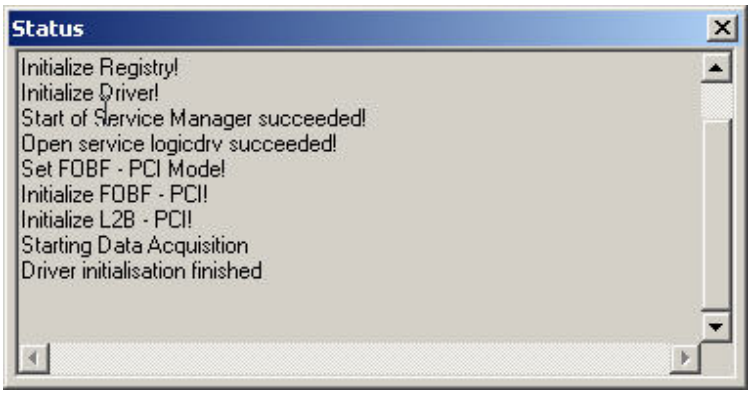


Fig. 5 State window ibaLogic-V3-Runtime

If you click the right mouse button at the runtime label at the task bar a context menu will be displayed like the following.

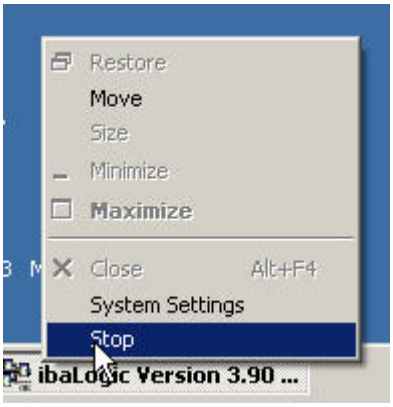
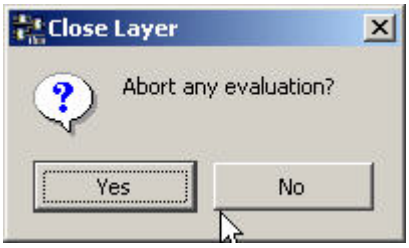


Fig. 6 Context menu ibaLogic-V3-Runtime

You may stop the runtime or open the system settings in this context menu.

If you call up the system settings, ibaLogic asks you to stop the running process of the runtime.



If you don't abort the evaluation only a view to the system settings is possible. If you abort any evaluation, you may configure the system settings.

The evaluation starts again after closing system settings.

You find a amplification of the system settings in chapter 2.5

2.1.3. Start ibaLogic with the command line

ibaLogic can also be started with the command line. Therewith it is possible to start ibaLogic with a batch file or with a Visual Studio application.

You can refer parameter by using the command line to start ibaLogic differently.

Syntax of the command line

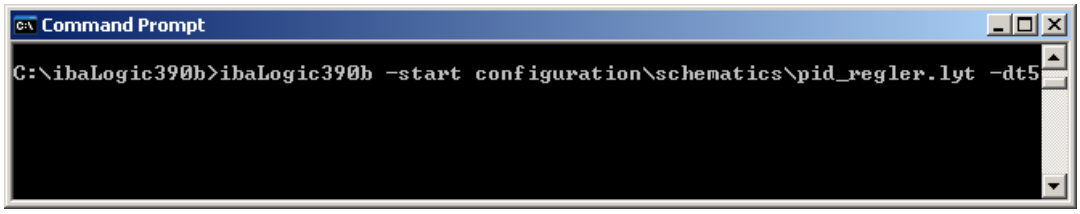


Fig. 7 Command line interpreter

C:\ibaLogicXXX>ibaLogicXXX –start

- **ibaLogic-V3-Runtime:** starts the runtime file automatically.
- **ibaLogic-V3:** starts ibaLogic-V3 with an empty layer.

C:\ibaLogicXXX>ibaLogicXXX –start configuration\schematics\Datei.lyt

- **ibaLogic-V3:** starts ibaLogic-V3 with a file.lyt and locked the layer.

C:\ibaLogicXXX>ibaLogicXXX –start -dt

You can preset a default value –dt for the base time. This is needfully if ibaLogic starts for the first time.

XXX=Version number

2.2 ibaLogic user interface

After start-up, ibaLogic shows a screen like the following:

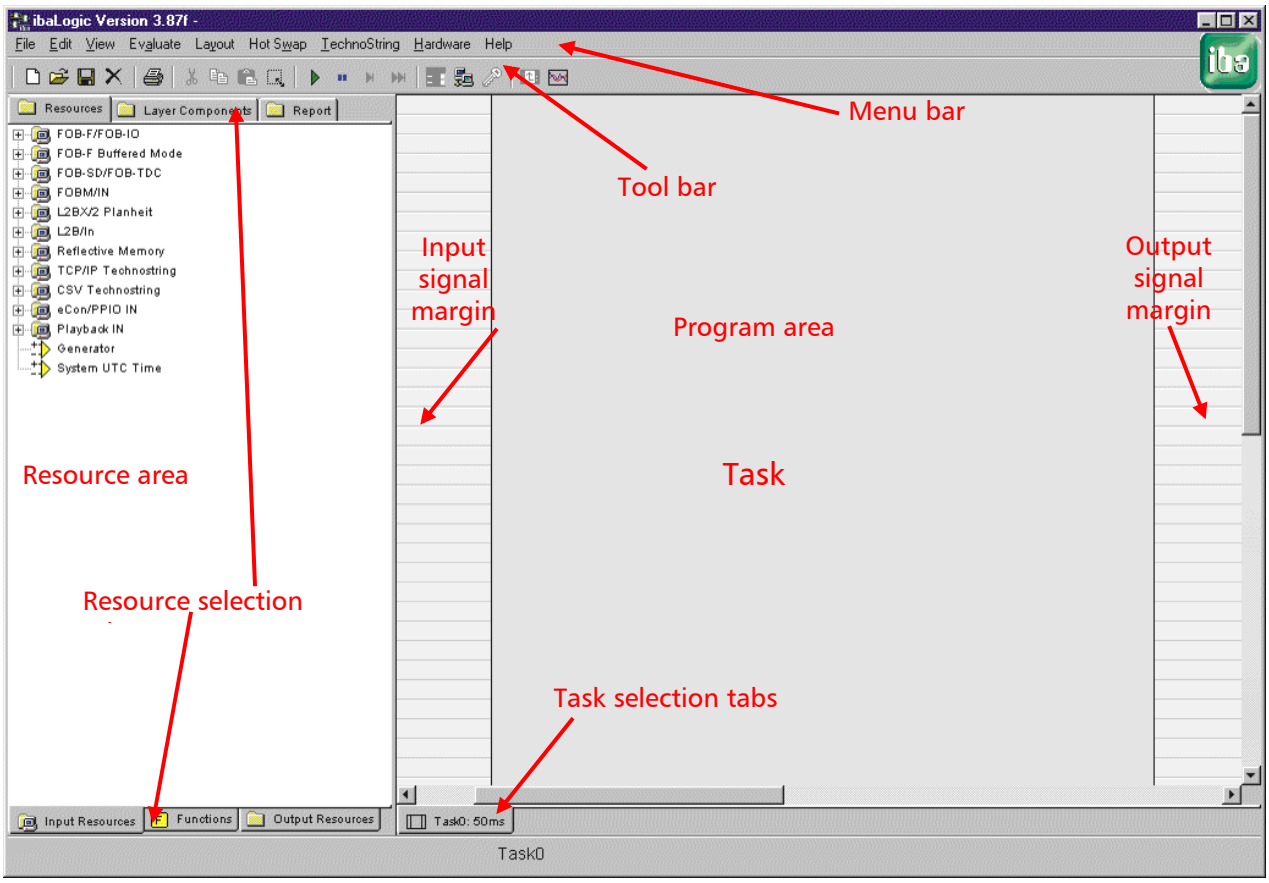


Fig. 8 ibaLogic standard screen

Like in many other Windows® applications, a menu bar (drop-down menus) and a tool bar with buttons for frequently used commands are located in the upper part of the screen. The commands of the menu and the tool bar are explained in the next chapter of this manual.

For the application ibaLogic uses two major areas. On the left side of the screen there is the *resource area*. This area is devided into three views, which can be selected by clicking on the tabs on top of the resource area: the *recources*, the *layer components* and the *report*. Once a view is selected, the corresponding options appear for further selection at the bottom of the resources area.

The resources are devided into three groups: *input resources*, *functions* (incl. function blocks) and *output resources*. The desired resource group can be selected by clicking on the *resource selection* tabs.

To use a resource (e.g. the analog input no. 1 of module no.1 on the FOB/FOB-F interface card) just click on the desired resource, hold the mouse button, drag the mouse over the desired part in the *task area*, i.e. *input signal margin*, *program area* or *output signal margin*, and leave the mouse button (drag and drop).

If you'd prefer to use the full screen for the task area, just hide the display of the resource area by choosing the menu ↪ *View* ↪ *none*.

The "*Layer Components*" section shows all resources and objects which are used in the current layout. For different requirements there are three different views, using a tree structure:

Under the tab *Hirarchy* you'll find for each task the resources distinguished by their types: *FB and Macros, inputs, outputs, off-task inputs, off-task outputs and intra-page (connectors*). In order to find a particular resource, just click on the resource name in the tree and the display of the function block diagram will jump to the corresponding spot and mark the resource.

Under the tab *Objects,* similiar to the hierarchy-view, all objects which are used in the layout are listed but in an order sorted alphabetically by object types. Going deeper in the tree structure leads to the tasks and final instances of these objects. A click on an object instance will switch the function block diagram to the corresponding spot.

Under the tab *Instances,* the view is alike the previous one but the objects are sorted alphabetically by instance names.

The "*Report*"-view provides two further options: *Evaluation order* and *Feedback-loops*.

The evaluation order of the functions is shown in a tree-structure as well. Below each task all related functions are listed corresponding to the evaluation order. The first function is evaluated at first, the last function at last. The knowledge about the evaluation order is important when troubleshooting complex and encapsulated programs. By clicking on the function name in the tree, the display switches automatically to the corresponding spot in the function block diagram and highlights the function block.

The "*Feedback-loop*"-view shows all feedback-loops, i.e. endless loops and unintended recursions which may cause problems if available. All functions which are part of such a loop will be displayed in the tree, sorted by tasks. To find the related functions, use the same method as described before.

The application programs created by the user are assigned to tasks. Each task has its own cycle time (period), e.g. 50 ms. The period of each task is shown in the task selection tabs. You can switch from one task to another by clicking on the task selection tabs. All tasks put together are a layout or a project which is stored both in a **.lyt*-file and in a "Structured Text" (**.txt*-) file.

As soon as ibaLogic is set to evaluation mode or to online mode the "Evaluation [%] display" appears in the lower left corner of the screen. This display shows the percentage of time spent for processing the tasks in relation to their defined period.

2.2.1. Tool bar

The ibaLogic tool bar consists of short cuts for commands as follows:

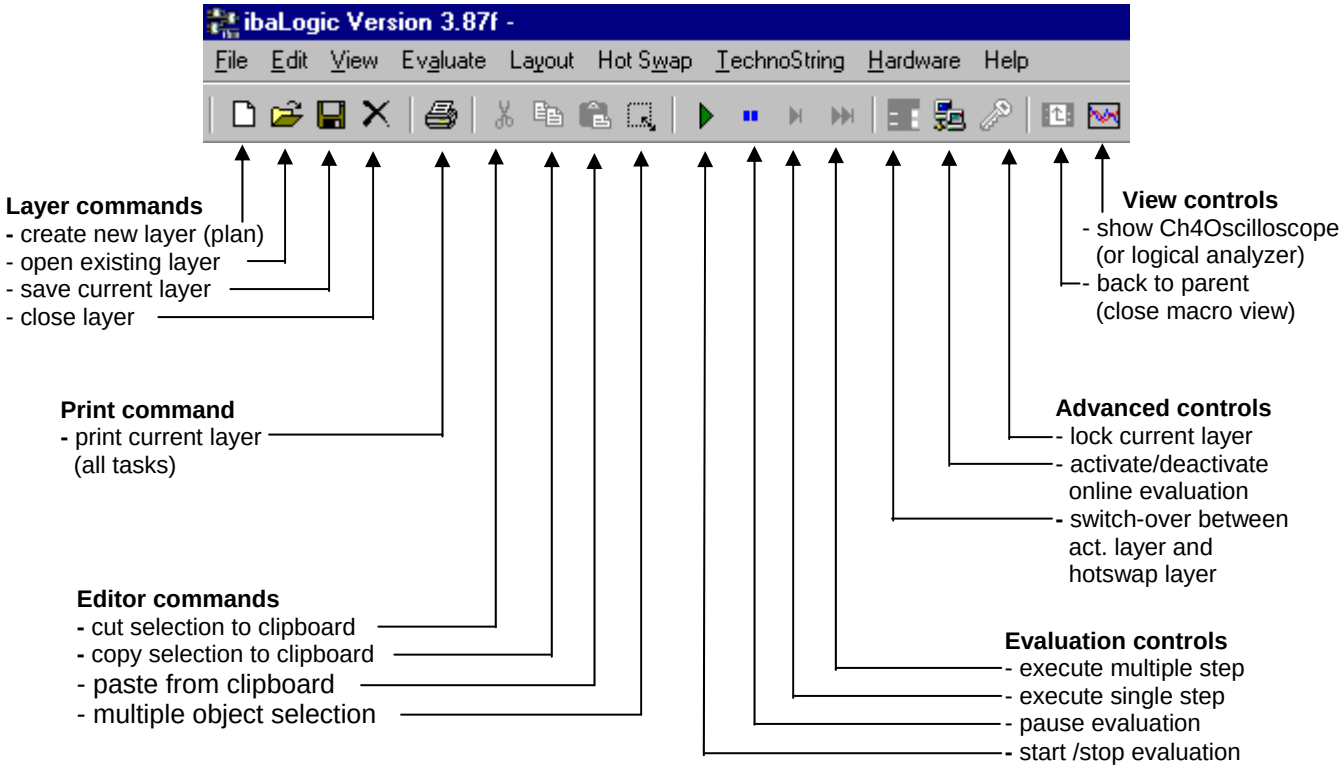


Fig. 9 Tool bar

2.2.2. Hot keys

Key combination	Function
<CTRL>+<A>	Open an existing layout (*.txt)
<CTRL>+<Backspace>	One level back (inside a macro, up)
<CTRL>+<C>	Copy marked object to the clipboard.
<CTRL>+<M>	Activate multiple object selection (followed by outlining the objects with the mouse).
<CTRL>+<N>	Create a new layout
<CTRL>+<O>	Open an existing layout (*.lyt)
<CTRL>+<P>	Print current layout
<CTRL>+<Q>	Stop evaluation
<CTRL>+<S>	Save current layout
<CTRL>+<V>	Paste contents from clipboard
<CTRL>+<X>	Cut marked object and put it on the clipboard
<Alt>+<ENTER>	Edit marked object
<Alt>+<I>	Single step for evaluation
<Alt>+<L>	Lock / release online layer
<Alt>+<M>	Multiple step for evaluation
<Alt>+<O>	Online / Offline-switching
<Alt>+<P>	Pause evaluation

Key combination	Function
<Alt>+<R>	Reset and restart evaluation
<Alt>+<S>	Start / Stop evaluation
<DEL>	Delete marked object

Table 2 Hot keys

2.2.3. Combinations of mouse keys and keyboard

LM = left mouse key RM= right mouse key

Keyboard	Mouse	Function
	LM (click)	Mark an object in program or resource area
<CTRL>+	LM (click)	Mark another object in program or resource area (successive); when marking objects which are linked to each other, the connection lines are marked too.
<Shift>+	LM (click)	Mark another object in program or resource area (successive); when marking objects which are linked to each other, the connection lines are marked too.
<Alt>+	LM (click)	Cut connection line and replace it by IntraPage-connector(s); mouse cursor must point on the line concerned.
	LM (doubleclick)	On function block: open function block On symbolic name: change name
	LM (hold)	Shift view on program area on the screen, when mouse pointer is placed in empty space (mouse pointer change its shape to cross pointer)
	LM (hold)	Selection of one or more objects in program area by outlining and shifting a marked object or object group
	LM (hold)	Changing route of connection lines, when mouse pointer shows cross-shape at line kinks
	LM (hold)	Extend the program area by another page on the right side or bottom side; the mouse pointer has to be placed on the far right or lowest margin of the program area, then it changes shape to a double pointer, then draw it over the border to the right resp. down.
	RM	Open a context menu, if available, e.g. in program area or on tabs in the task selection bar.

Table 3 Combinations of keyboard and mouse operation

2.3 ibaLogic menu bar

2.3.1. "File" menu

Fig. 10 "File" menu

❑ ***File commands***

- *New*: Create a new layout "Project"
- *Open*: Open an existing layout, (*.lyt)-file
- *Open ASCII*: Open ASCII file (*.txt) (**S**tructured **T**ext)
- *Open DLL*: Open an (imported) DLL-function
- *Save*: Save the current layout as *.lyt-file
- *Save ASCII*: Save the current layout as **S**tructured **T**ext (ST) in an ASCII-file (*.txt)
- *Save As*: Save the current layout in *.lyt- and *.txt-file under new name Remark: ASCII-Structured Text-files are independent from ibaLogic software versions and should be used and stored for backup.
- *Close*: Close the current layout.

❑ ***Password and printer commands***

- *Change Password*: Enter or change the online password. After activation of password, modifying, saving and closing the project are locked by correct password input. Thus, hot-swap layers can be protected from switch-over.
- *Print*: Opens a window with a variety of printing options in order to specify whether to print the entire layout (all tasks) or just a choice of objects.
- *Page Setup*: Setup of page layout, e.g. page size, margins etc.

❑ ***Settings***

- *Program settings*: Open dialog window for program settings, see section 2.4.
- *System settings*: Open dialog window for system settings, see section 2.5.
- *PCI configuration*: Open dialog window for PCI configuration, see section 2.6.
- *ISA-configuration*: Open dialog window for ISA configuration, (not available with Windows XP)
- *Restart driver*: Restart the communication drivers
- *Exit*: Close and exit ibaLogic

2.3.2. "Edit" menu

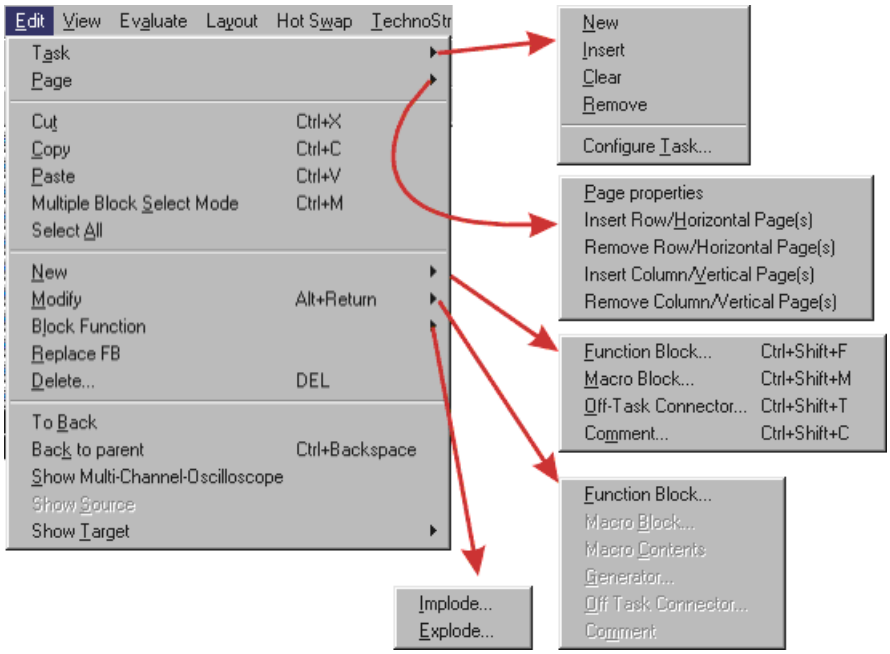


Fig. 11 "Edit" menu

❑ ***Task commands:***

- *New*: Create a new Task
- *Insert*: Insert a new Task (ahead of the current task)
- *Clear*: Delete contents of a task
- *Remove*: Delete the selected task completely
- *Configure Task*...: Task configuration: definition of task name, cycle time and size of program area.

❑ ***Page commands:***

- *Page properties*: Open dialog window for entering information to be printed on the pages.
- *Insert or Remove Row / Horizontal Page(s)*: Insert a new page or row of pages on top of the current page.
- *Insert or Remove Column / Vertical Page(s)*: Insert a new page or column of pages left from the current page.

❑ ***Function block commands (1):***

- *Cut:* Cut out function block or multiple selction
- *Copy:* Copy selected elements
- *Paste:* Insert selected elements (cut or copied)
- *Multiple Block Select Mode:* Alter the cursor function to „rubber band" for selection of a group of function blocks, lines and comments

❑ ***Function block commands (2):***

- *New:* A further submenu opens for creating a new function block, macro block, off-task connector or comment.
- *Modify:* A further submenu opens for modification of the above mentioned blocks and elements. (Element to be modified must be selected)
- *Block Function*:
  Implode: *:* Build a macro by combination of the selected function blocks, lines and comments
  Explode: : Break down a macro block into its components and insert them

- *Replace FB*: Open a dialog window for replacing one function block by another. Choise of reference to one instance or all instances of the FB.
- *Delete*: Delete selected elements

You may get the menu "Edit" also by clicking the right mouse key when pointing into the program area (contextmenu).

❑ ***Navigation***

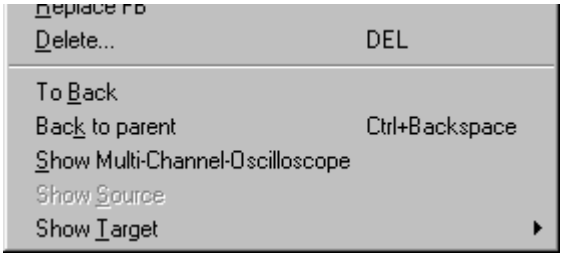


Fig. 12 "Edit" menu, navigation commands

- *To Back:* Put marked object in the background (graphically)
- *Back to parent*
  Switch back to an upper program level, i.e. leave the macro level.
- *Show Multi-Channel-Oscilloscope:* Open a window for display of the selected multi-channel-oscilloscope or logic analyzer
- *Show Source:* Show the connection to the source (task) of a selected off-task connector (input).
- *Show Target:* Show the connection(s) to one or more targets of a selected off-task connector (output).

2.3.3. "View" menu

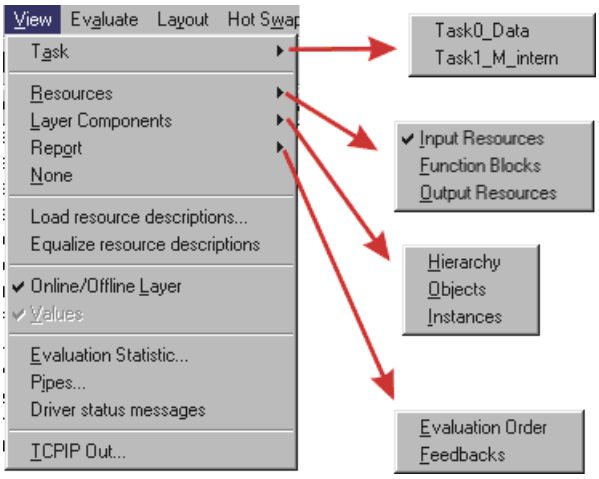


Fig. 13 "View" menu

❑ ***Task commands***

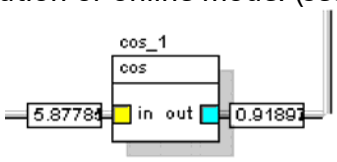
- *Task*: Selection of available tasks (e.g. 0..1)

❑ ***Resource selection commands***

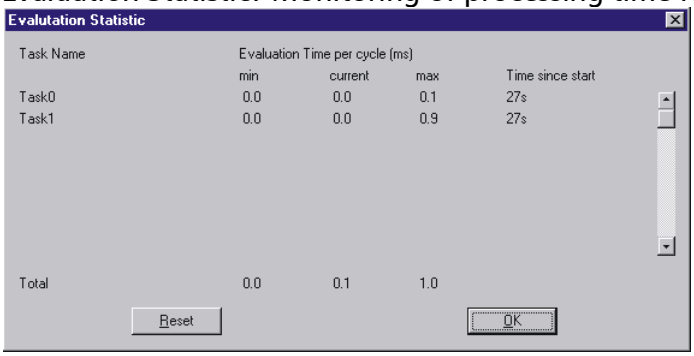
- *Resources*
  Input Resources: Open the directory of input resources
  Function Blocks: Open the catalogue of functions and function blocks
  Output Resources: Open the directory of output resources
- *Layer Components*
  Hierarchy: Open a tree structure which shows the objects, used in the project (layout), arranged according to their hierarchy, i.e. by tasks. Mouseclick on an object in the tree will lead to the object in the function block diagram.
  Objects: Open a tree structure which shows all objects and instances in the project (layout), arranged in an order according to their *type*. Opening the tree branches will show where these instances are used. A further click on the taskname will lead to the object in the corresponding task and function block diagram.
  Instances: View similar to previous but sorted according to their *instance names*.
- Report:
  Evaluation Order: Show the evaluation order of the tasks and their objects. (top-down).
  Feedbacks: Show "endless loops" if present. Mouseclick on shown objects will lead to the corresponding spot in the function block diagram.
- *None*: Close the resource area on the screen completely, so that the screen is only used for program area.
- *Load resource descriptions*: Load modified descriptions of I/O-resources, e.g. I/O resources which had been renamed by an external editor and saved as CSV-files. (For creation of such CSV-files, just select the desired resource with the right mouse button and confirm export.)
- *Equalize resource descriptions*: Signal names from the function block diagram can be used for resource description. Vice versa the resource description can be used in the diagram.

❑ ***Layer control***

- *Online/Offline Layer*: Switch-over between online- and offline layer in "Hot-Swap" mode.
- *Values*: Display of current signal values of function blocks, task in- and outputs in evaluation or online mode. (see example below: "Values on")

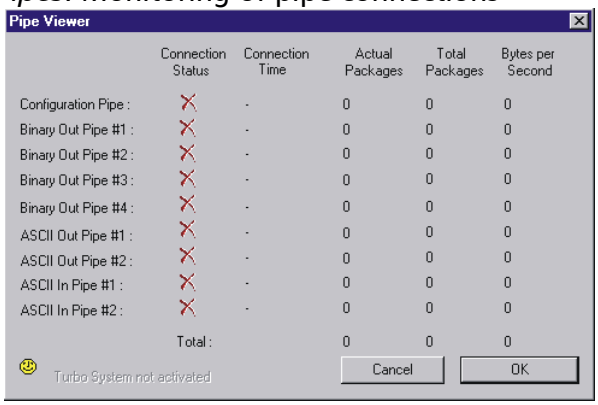


- *Evaluation Statistic:* Monitoring of processing time for each task, see below



Shows an overview about the different tasks with information about task name, processing time per cycle (in ms) with minimum, maximum and actual value, the total of these values and the overall runtime.

- *Pipes*: Monitoring of pipe connections



The "Pipe Viewer" shows an overview of the current status of configured pipe connections ("pipes").

↗ *See also 5.1.8*

- *Driver status messages*: Open a dialog window with status messages about ibaLogic, e.g. restart of drivers, initialization of registry etc.
- TCPIP Out: Open a dialog window with an overview of the current status of configured TCP/IP connections.

2.3.4. "Evaluate" menu

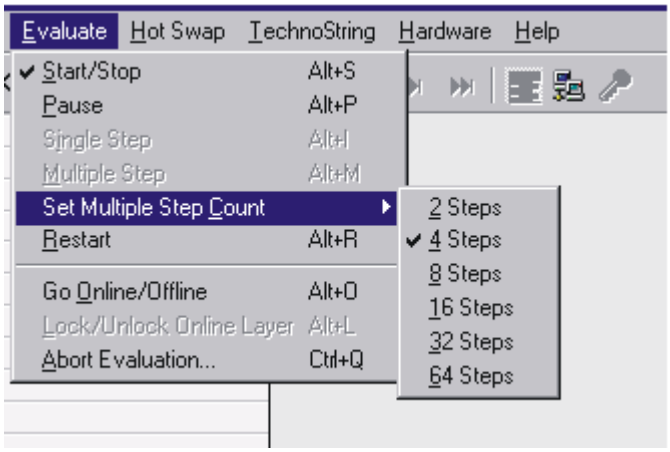


Fig. 14 "Evaluate" menu

❑ ***Control of evaluation mode***

- *Start/Stop*: Start/Stop the offline evaluation of all tasks (evaluation mode)
- *Pause*: Pause or continue the evaluation mode
- *Single Step*: Evaluation of one program cycle (all tasks)
- *Multiple Step*: Evaluation of multiple program cycles
- *Set Multiple Step Count*: Setting the number of steps (2..64) for "*Multiple step*"
- *Restart*: Restart all tasks

❑ ***Control of online / offline mode***

- *Go Online/Offline*: Switch between online and offline mode. The activated online mode is indicated by purple background color on the screen.
- *Lock/Unlock Online Layer*: Locking of the current online layer with input of a password (if a password is defined) will prevent switching to offline mode and modification of this layer. The online layer must be locked in order to create a hot-swap layer.
- *Abort Evaluation*: After confirming the command, online mode resp. evaluation mode will be interrupted immediately.

2.3.5. "Layout" menu

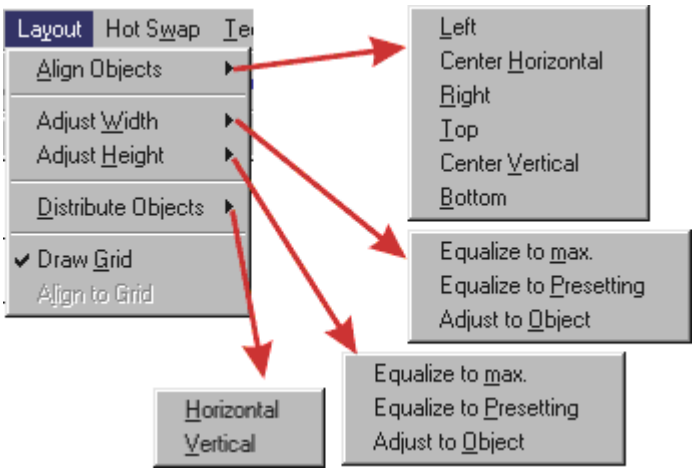


Fig. 15 "Layout" menu

❑ *Layout commands*

The layout commands refer to the representation of objects in the program area of ibaLogic, such as function blocks, off-task-connectors or comments. The objects concerned should be marked first.

- *Align Objects*: According to the submenu the marked objects will be aligned along a common line. The terms Left, Right, Top and Bottom refer to the object borders, the terms Center Horizontal and Center Vertical refer to the (virtual) center lines of the objects.
- *Adjust Width, Adjust Height*: The corresponding submenus offer different kinds of adjustments
  Equalize to max.: More than one object should be marked. This command adjusts the width resp. height of all marked objects to the widest resp. highest object in the group.
  Equalize to Presetting: One or more objects may be marked. The command adjusts the width resp. height of the marked objects according to the presettings given in the menu ↪*File* ↪*Program Settings* ↪*Edit*.
  The limit in terms of downscaling is the full representation or legibility of the entire contents of an object, e.g. all input and output connectors of a function block.
  Adjust to Object: This command adjusts the width of an marked object according to the full legibility of its contents. In case of a height adjustment the preset distance between connectors of a function block, given in the menu ↪*File* ↪*Program Settings* ↪*Edit,* is taken into account.
- *Distribute Objects*: At least three objects should be marked. According to the preset, given in the menu ↪*File* ↪*Program Settings* ↪*Edit* the marked objects will be distributed in vertical or horizontal direction with an even distance referring to their left or top edge or with an even gap between two objects.

2.3.6. "Hot Swap" menu

Fig. 16 "Hot Swap" menu

❑ ***Hot Swap control***

- *Create*: By means of a hot-swap layer it's possible to create a copy of a current layer which is running in online mode, to modify it and to switch over during operation („hot"). In order to create a hot-swap layer the following steps have to be made:
  1.) Switch-over to online mode „Go Online"
  2.) Lock Online Layer (key button)
  3.) Create hot-swap layer (menu ↪*Hot Swap* ↪*Create*). This command will create a copy of the contents of the current online layer without leaving the online mode. The copied hot swap layer is now ready for modification, but without affecting the online execution.
- *Apply to Online Layer*: The modified hot-swap layer will be set online during operation.
  Remark: When created, the hot swap layer acquires the "memory", i.e. the values and signal states, of the online layer. When applied to online layer, the hot swap layer acquires the memory of the online layer again for the program elements which are already existing. For new function blocks (with memory) the values and signal states are taken from the hot swap layer. This manner ensures that changes of values and signals in the online layer, e.g. operator commands via OPC, don't get lost.
- *Close*: Close and leave the hot swap layer. Changes will get lost unless the layer have been switched online or the changes have been saved.

2.3.7. "Technostring" menu

Fig. 17 "Technostring" menu

❑ ***TechnoString***

TCP/IP: Open the window "TCP/IP Technostring" as shown below which offers the possibility to assign technostring variables to input variables.

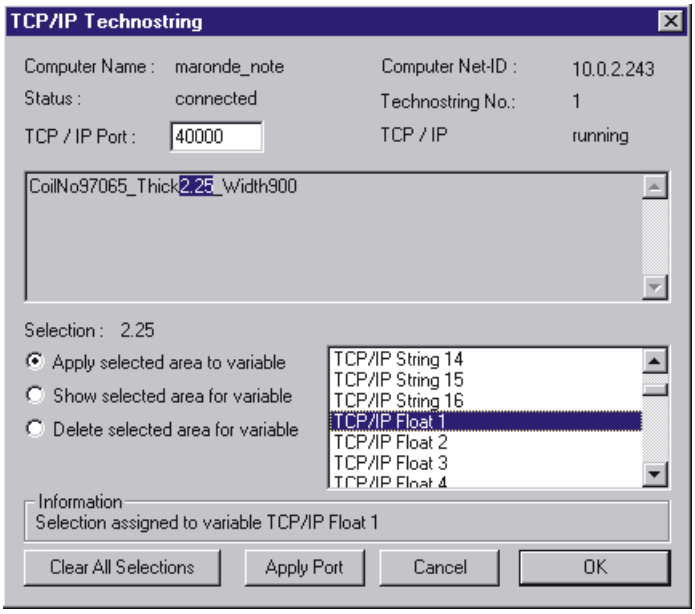


Fig. 18 TCP/IP Technostring, dialog window

In this window one can parse an incoming TCP/IP technostring of any structure. Its contents can be assigned with reference to the character index either to string variables (TCP/IP STRING 1...16) or to float variables (TCP/IP FLOAT 1...96).
In order to assign a part of a technostring to a variable please follow these steps:

1. As a precondition the TCP/IP operation must be enabled for ibaLogic (menu ↪File ↪System Settings ↪Miscellaneous)
2. In the field "*TCP/IP Port*" enter the correct port number. This port number must be the same like in the systemg which sends the technostring. (see box below)
3. Using another (remote) system send a technostring to the ibaLogic computer by means of the test program *TCPIP Test.exe*. The technostring should appear in the dialog window above (Fig. 18).

The decoding of the technostring is strictly index orientated, i.e. the incoming string must always have the same format.(Peril when suppressing leading zeros!)

4. Check the box "*Apply selected area to variable*"
5. Then mark the desired area in the technostring field by using the mouse (hold left button), in this example "2.25". The selected characters are repeated behind the term "Selection:"

6 Then select the desired variable in the variable field, e.g. TCP/IP Float 1, and doubleclick on variable name or click OK.

7 Repeat steps 4 to 6 for other variables if required.

8 In order to check the correct assignment of technostring and variables, just check the box "*Show selected area for variable*" and select one of the recently assigned variables. The corresponding part of the technostring will be highlighted.

9 When you've finished, click on *OK* to close the dialog window.

When exiting the dialog window the ASCII-file *iba_tcp.cfg* is created in the folder *configuration* in order to save the assignments.

ibaLogic uses the default value 1500 as TCP/IP port number for technostring communication (reception), unless another port number is saved in a file iba_tcp.cfg.

If a different port number has to be used for technostring communication because this port is used for other kinds of data exchange, then enter a new port number in the dialog window above (Fig. 18) and click OK in order to create the file iba_tcp.cfg.

If this file is available at startup of ibaLogic, the included port number will be used.

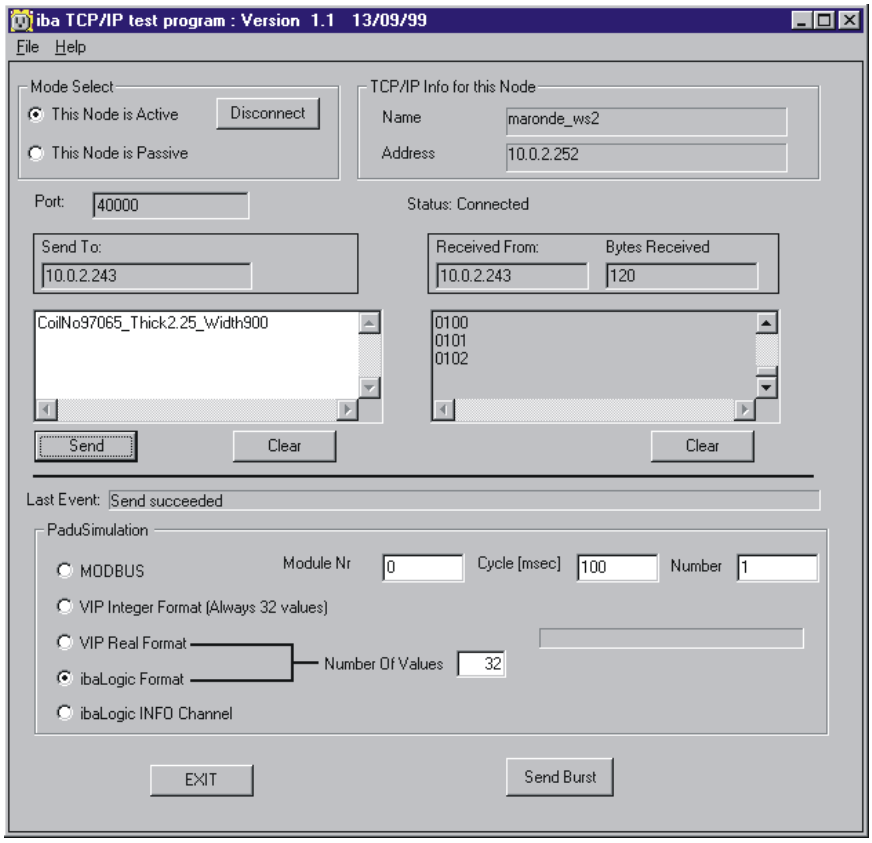
Remark: In order to verify the proper work of the technostring function in the network, a test program tcpip.exe is in the scope of supply of iba.

Simply enter the network address (name and IP-address) of the lokal PC and select a port number.

Enter the IP-address of the target-PC (running with ibaLogic) and type in a text message.

Then enter the same port number in the mask as shown above on the target PC.

Set the lokal PC on "this Node is Active", click on "Connect" and then on "Send". The message should appear in the field as shown above on the ibaLogic-PC.



2.3.8. "Hardware" menu

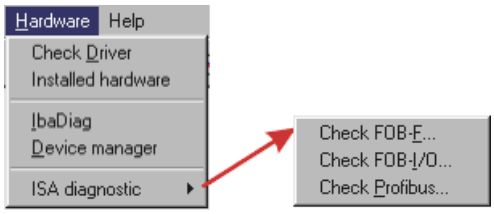
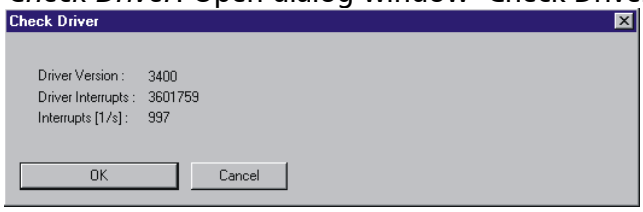


Fig. 19 "Hardware" menu

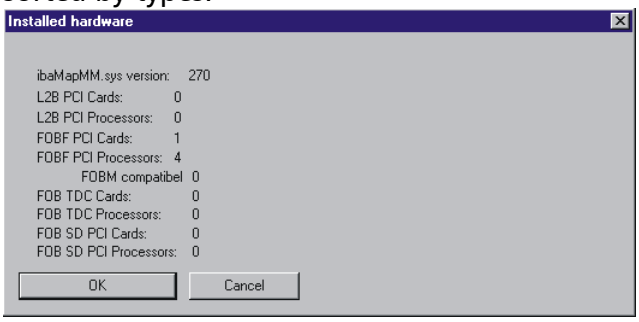
❑ ***Hardware***

- *Check Driver*: Open dialog window "Check Driver" (see below).



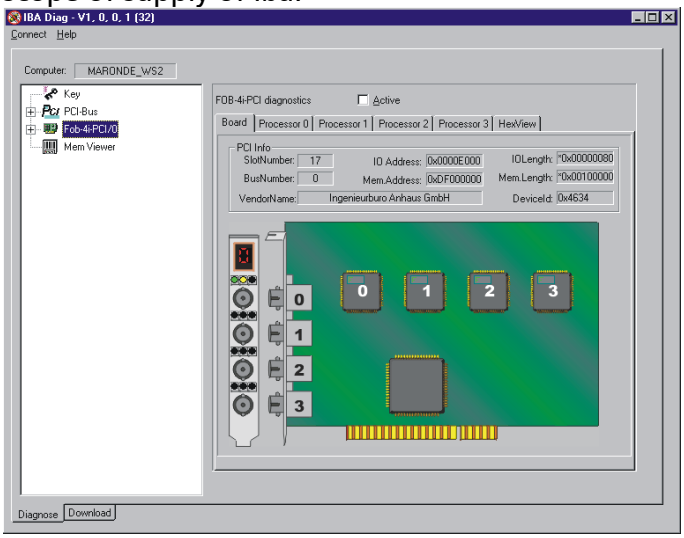
If installed properly, "*Interrupts [1/s]:*" should show approx. 1000. (may vary). Exception: FOB 4i PCI in asynchronous mode.

- *Installed Hardware*: The system detects automatically the (iba) hardware components which are installed in the computer and shows the number, sorted by types.



The example above shows that a FOB 4i PCI card is installed (one card and four processors).

- *IbaDiag*: Start the diagnostic program *ibadiag.exe*, which is part of the scope of supply of iba.

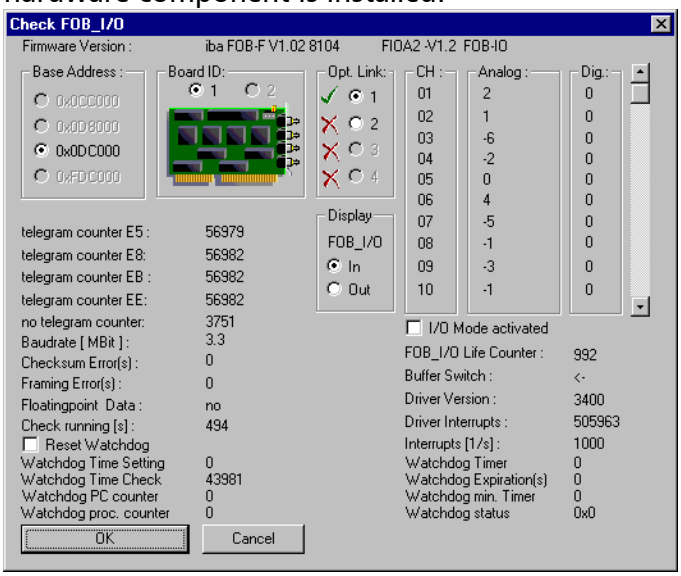


Example for the display of ibaDiag.
ibaDiag can also be started independently from ibaLogic on a PC.
Beside the detailed view on the cards ibaDiag also provides a lot of information about the PCI-bus and the connected components.

For more detailed information about the program ibaDiag please refer to corresponding manual ***sw_man_ibaDiag_en_a4.pdf*** *(or ..._LTR.pdf for letter format).*

- *Device manager*: This menu command works only with Winows XP. It calls the Windows device manager for display of drivers and hardware settings. If iba I/O cards are installed in the PC you'll find a branch which is called *iba Devices* in the tree structure of the device manager window. Open this branch and you'll find the installed iba cards. A doubleclick on the card icon opens the information dialog.

- *ISA-diagnostics + submenu*: Open the dialog window for FOB-F, FOB-I/O (see example below) or Profibus via the submenu if the corresponding ISA-hardware component is installed.



Example of an ISA-display.
New systems of iba will be equipped with PCI-cards only, because the ISA-bus technology is in a dead end and not supported anymore in the PC industry. In case of use of ISA cards we'd like to refer to Version 2 of the ibaLogic manual.

2.3.9. "Help" menu

Fig. 20 "Help" menu

- *Contents*: Open Online-help function (requires help file)
- *About...*: Display of current ibaLogic software version

2.4 Program settings

2.4.1. Menu ↪File ↪Program Settings ↪General

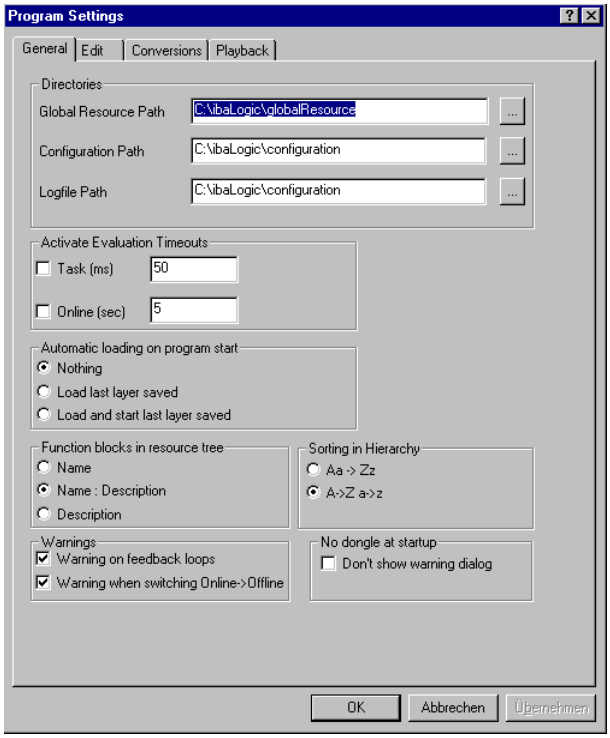


Fig. 21 Program settings, general

❑ ***Directories***

- *Globale Resource Path*: Pathname for global resources, i.e. libraries, macros, function blocks (FBs) and DLLs created by the user.
- *Configuration Path*: Pathname for Configuration with DLLs, FBs, macros, libraries and functionblock diagrams (projects *.lyt / Structured Text *.txt).
- *Logfile Path:* Pathname for the logfile, which is generated by ibaLogic.

❑ ***Activate Evaluation Timeouts:***

The evaluation or online mode will be interrupted as soon as the adjusted evaluation timeout(of the task (e.g. 50 ms) or ibaLogic (e.g. 5 s) has passed (watchdogs). This function interrupts unintended continuous program loops, created by the user, or reactivates ibaLogic in case of an major error.

The evaluation may also be aborted if the values entered for evaluation timeouts are too low.

❑ ***Automatic loading on program start:***

Definition of startup behaviour of ibaLogic; this is to activate the automatic start-up or to shortcut the continuation of engineering.

❑ ***Function blocks in resource tree:***

Select how the functions should be displayed in the resource tree: with name or with description or both.

❑ ***Sorting in hierarchy:***

Regel für die alphabetische Sortierung der Objekte in der Ansicht "Layer Komponenten" / "Hierarchie" (Ressourcenbereich); ohne oder mit Unterscheidung der Groß-und Kleinschreibung.

❑ ***Warnings:***

- *Warning on feedback loops:* This option enables the "endless loop"-detection of ibaLogic which informs the user already during the programming about feedback loops.
- *Warning when switching Online -> Offline:* This warning is to avoid an unintended switch-over to offline mode when the process is running.

❑ ***No dongle at startup:***

If this box is not checked a message will appear during startup of ibaLogic in case that no dongle has been detected. The dialog window offers some alternativs for starting ibaLogic even without dongle, e.g. demo mode or eCon mode.

2.4.2. Menu ↪File ↪Program Settings ↪Edit

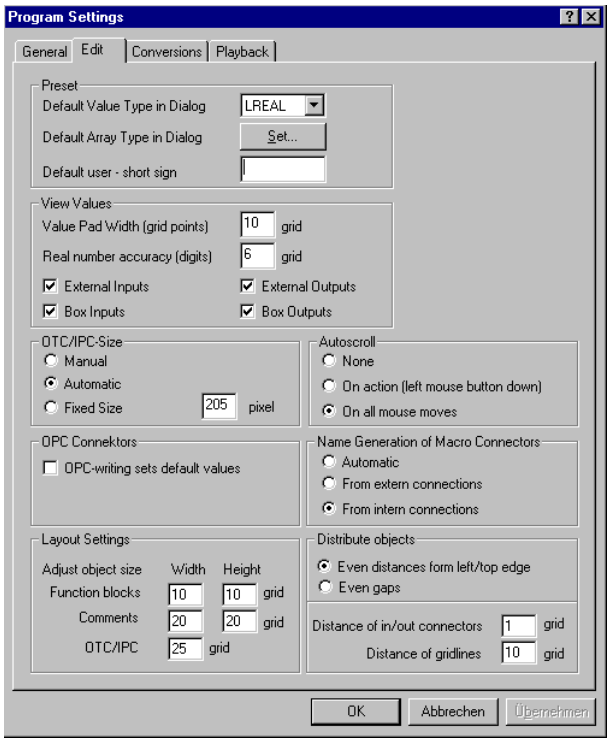
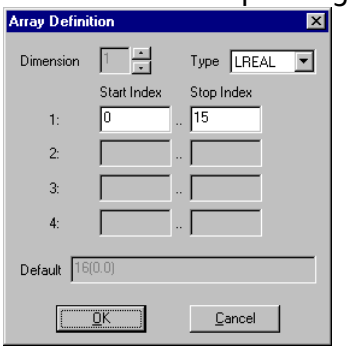


Fig. 22 Program settings, edit

❑ ***Preset***

- *Default Value Type in Dialog*: Predefined datatype, e.g. LREAL in FBs
- *Default Array Tape in Dialog*: Predefined arraytype, e.g. 2-dimensional LREAL
- *Button "Set"*: setup dialog for default arraytype



With changing the dimension, using the up/down buttons, one- to four-dimensional, the corresponding index fields below can be activated.
Start index and stop index, resp. their difference, decribe the number of array elements for each dimension.
The display field "Default" shows the number of array elements and the default values. The example above shows an one-dimesional array with 16 elements. The value of each element is 0.0.

- *Default user – short sign*: The user may enter his initials here. They are used for example, in the printouts of the layout.

❑ ***View Values:***

By using menu ↪*View* ↪ *Values* it's possible to view the actual values of signals in FBs and of in- and outputs, if selected by the check boxes below.

- *Value Pad Width [grid points]*: Adjustment of number of digits for value display at the FBs in evaluation and online mode, given in grid points. You may get a better idea of the size of a grind point when you switch on the grid display in the program area (menu ↪*Layout* ↪*Draw Grid*).
- *Real number accuracy (digits)*: Number of decimal places for Real and LReal values;
- *Check boxes for input and output types*: Selection of types to be displayed;

❑ ***OTC/IPC-Size:***

Selection of the size of graphical representation for new off-task and intra-page connectors. For example, in mode "Automatic" the connector size will always be adjusted to the name of the connector.

❑ ***Autoscroll:***

- *None*: Autoscroll is switched off, i.e. the navigation in the program area occurs by pushing the left mouse button and moving the cursor.
- *On action (left mouse button down):* Navigation either by pushing the left mouse button (s.a.) or automatically when shifting FBs or drawing connection lines.
- *On all mouse moves*: Autoscroll is switched on, i.e. navigation in the program area occurs every time the cursor is close to the window margin.

❑ ***OPC-Connectors:***

- *OPC-writing sets default values*: If this option is selected, the default value of an OPC-connector may be overwritten by an OPC-client. Using this feature each new value, e.g. manually entered via an HMI system, is taken for the new default value by the OPC-connector. Thus, the OPC-connector takes the latest actual value as default in case of a program restart. If this option is not selected always the same default values as engineered will be used. The use of this option is only relevant for OPC-connectors with an activ OPC→ ibaLogic flag.
- *Use new OPC Server version:* The usage of the new OPC Server is strongly recommended.

❑ ***Name Generation of Macro Connectors:***

Each connector of a function block has a name. When combining several function blocks to one macro block (implode) the new input and output connectors of the new macro block are created at the cuts of the connection lines between the objects inside and outside the macro block. Depending of the choice of this option the input and output connectors of the macro block will be named automatically or according to the connctor names of the inner, resp. outer function blocks.

❑ ***Layout Settings***

The layout settings are used for the functions "Adjust Width" and "Adjust Height" in the menu ↪*Layout*. The values are given in grid points as unit.

- Function blocks: Presets for the size of function blocks
- Comments: Preset s for the size of comment fields
- OTC/IPC: Preset for width of off-task and intra-page connectors

↗ *See also chapter 2.3.5*

❑ ***Distribute objects:***

- *Even distances from left / top edge*: Marked objects (at least three) will be positioned in even distances with reference to their top or left edge when the function "Distribute objects" in the menü ↪*Layout* is used. Overlapping of objects is may occur.
- *Even gaps*: Marked objects (at least three) will be positioned with even distances between them when the function "Distribute objects" in the menü ↪*Layout* is used.

Other settings:

- *Distance of in/out connectors*: A minimal distance between two function block input or output connectors can be set. The setting will be applied when using the command "Adjust to object" in the menu ↪*Layout*↪*Adjust Height*.
- *Distance of grid lines*: The distance of grid lines given in grid points as unit may be entered here. In order to see the grid just choose menu ↪*Layout* ↪*Show grid*.

↗ *See also chapter 2.3.5*

2.4.3. Menu ↪File ↪Programm Settings ↪Conversions

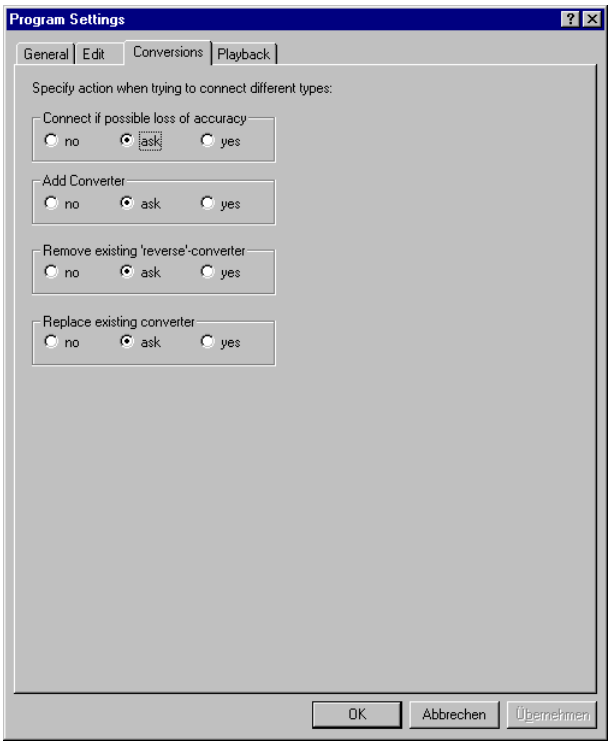


Fig. 23 Program settings, conversions

The selection of these options will define the actions ibaLogic performs automatically in an attempt of connecting variables of different datatypes.

❑ ***Choice:***

- Connect if possible but loss of accuracy
- Add datatype-converting function block (converter) to the connection
- Remove existing 'reverse'-converter
- Replace existing converter

Default setting is "ask", i.e. in case of a datatype conflict when making a connection a dialog window will pop up urging the user to confirm or reject the action.

2.4.4. Menu ↪Files ↪Program Settings ↪Playback

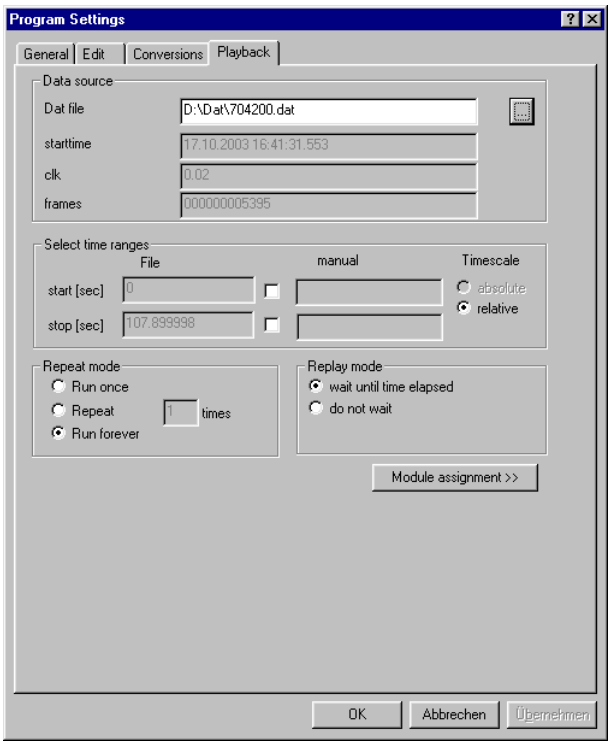


Fig. 24 Program settings, playback

❑ ***Data source:***

- *Dat file*: Path and name of the data file (+.dat) which is supposed to be used as signal source. Please use the button ... to browse if needed. If a valid file has been found, the signifcant information is displayed in the appropiate fields (starttime, sample time and number of samples).

❑ ***Select time ranges:***

This option allows to limit the range of time in the data file which should be replayed in playback mode. For manual entries of start- and/or stoptime please check the corresponding boxes.

❑ ***Repeat mode:***

Choice of how often the data should be replayed.

❑ ***Replay mode:***

In order to reach a realistic replay it is necessary that the task-cycletime of ibaLogic is equal or smaller than the sample rate of the recorded data. When in playback mode, ibaLogic acquires a new sample from the data file in each task if the cycletime equals the sampletime of the recorded data. If the cycletime of ibaLogic is shorter than the sampletime of the data, the selection of the replay mode has the following results:

- *wait until time elapsed*: after reading one sample ibaLogic waits until the sampletime has elapsed before acquiring a new sample from the data file. (example: ibaLogic cycletime = 5 ms, sampletime in data file = 20 ms → ibaLogic acquires new samples every four cycles, for three cycles the same value is used.)

- *do not wait*: ibaLogic acquires a new sample in every cycle. As a result the playback looks like a time-lapse shot.

If the ibaLogic cycletime is longer than the sample time of the recorded data signals may "get lost" because ibaLogic takes the actual value in each cycle with reference to the correct time from the data file. The choice of "waiting" or not is irrelevant.

❑ ***Button*** **"Module assignment"**

Open the dialog window for assigning the input signals.

↗ *See also chapter* 3.6.4 for further information.

2.5 System settings

2.5.1. Menu ↪File ↪System settings ↪General

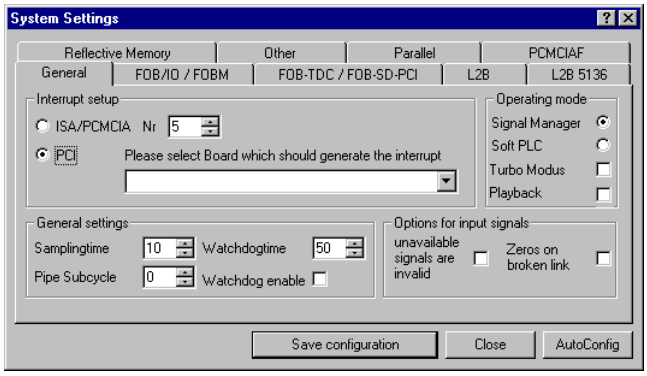


Fig. 25 System settings, general

❑ ***Interrupt setup:***

Selection of interrupt for ISA-boards or of the PCI-board, which is supposed to generate the interrupt.

❑ ***Operating mode:***

Selection of the operating modes of ibaLogic

- *Signal Manager* mode: The Signal Manager mode ensures that ibaLogic won't miss any incoming sample even if single tasks have been obstructed, i.e. "Evaluation [%]:" has been > 100 %.
  see chapter 3.6.1.
- *Soft PLC mode*: The Soft-PLC Mode which is suited for control and regulation tasks ensures that only the freshest signal values are processed.
  see chapter 3.6.2.
- *Turbo Modus*: Only to be used on PCs with double processor; if enabled one processor will exclusively be used for ibaLogic evaluation.
  see chapter 3.6.3.
- *Playback*: A data file of iba's *.dat-format which had been recorded before by ibaPDA, ibaScope or even ibaLogic, serves as a signal source;
  see chapter 3.6.4.

❑ ***General settings***

- *Samplingtime*: Setting of the basic cycletime for ibaLogic layouts. It should be shorter than the shortest task-cycletime used.
- *Watchdogtime*: Setting of the watchtime for the watchdog function. If the watchdog function is enabled (checkmark in the box) ibaLogic sends periodically watchdog telegrams to the related iba PC-cards. These telegrams should be sent by ibaLogic to the cards always within the watchdog time, like a trigger. The supervision of this process is done by the PC-cards, which "know" the time setting. If the watchdog telegram, i.e. the trigger, is not sent within the watchdog time, the cards lock the outputs on the fiber-optical side and reset them to zero (supported only by FOB IO, FOB 4i/4o [FOB-F]).

- *Pipe Subcycle*: A factor (integer) may be entered in this field. This factor refers only to the transmission rate of QDA-pipes (see also chapter 5.2.6). The pipe subcycle controls the transmission cycle of the QDA-pipes by using a multiple of the ibaLogic samplingtime (above). The use of this factor is only reasonable if the QDA-pipes must not be processed within the samplingtime. Thus the processor load may be reduced.

❑ ***Options for input signals***

- *unavailable signals are invalid*: Input resources of iba PC-cards (FOB IO, FOB 4i, L2B x/8 etc.) will be marked as invalid with a red frameline if the related card is not installed in the PC resp. unavailable.
- *zeros on broken link*: In case of a broken (optical) link to the input cards this option will cause the firmware of the cards to send zeros instead of the last value for the related input signals.
  ↗ See also chapter 3.7.1

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.2. Menu ↪File ↪System settings ↪Other

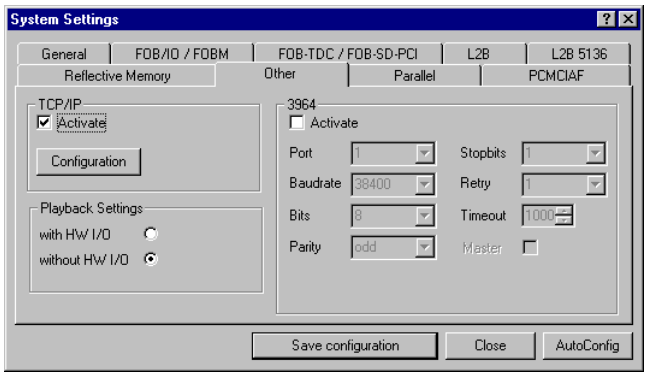


Fig. 26 System settings, other

This dialog is used for selection of other types of links for input and output signals.

❑ ***TCP/IP:***

Activate / inactivate the TCP/IP link as a source of data. TCP/IP must be activated (checkmark) for inputs/outputs via ABB VIP or Modbus (TCP/IP), for usage of technostring, for working with dlls which use TCP/IP communication and for usage of the function block "TCPIP_SendRecv. By clicking on the button "Configuration" the dialog for TCP/IP settings opens. The dialog "TCPIP settings" is used to make the required settings for connections over TCP/IP, see also chapter 2.6.6

❑ ***3964***

Activate / *inactivate* a serial link, e.g. of type 3964 R (DUST). The setting of the interface parameters should be done according to the target system. For a communication over 3964 there are dedicated function blocks available in ibaLogic.

❑ ***Playback Settings***

With or without HW I/O, i.e. with or without using the hardware inputs and outputs during playback operation. This feature allows to extend the range of applications for the playback mode. When "with" has been selected, data from a data file may be processed together with hardware signals. In order to avoid an overlapping of playback signals and hardware signals, special playback input resources are provided. See also chapter 3.6.4

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.3. Menu ↪File ↪System settings ↪Parallel

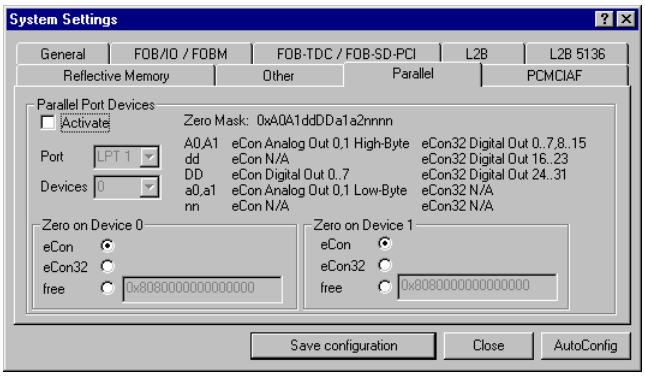


Fig. 27 System settings, Parallel

2

❑ ***Parallel***

- *Activate:* Activate / inactivate the parallel inetrface of the PC (printer port, lpt). This interface can be used for input and output of signals by connecting the **eCon**- and **eCon32**-devices from iba to it. Parallelschnittstelle des PCs (Druckerschnittstelle, LPT). This fuction is also available without a dongle.
- *Port:* From a pick-list choose the interface which is connected to the **eCon**-device. The BIOS of the PC must be set to bidirectional or EPP mode for this port!
- *Devices:* From the pick-list choose whether one or two **eCon**-devices should be used.

0	...if only one **eCon** is connected or ...if two **eCons** are connected but only the first one to be used
1	...if two **eCons** are connected but only the second one to be used.
0&1	...if two **eCons** are connected an both to be used.

❑ ***Zero on Device 0 / Device 1***

Check the radio buttons according to type of **eCon**-device(s) used at first and/or second position. Predefined zero masks are activated depending on the selection. The zero masks are used in order to reset all outputs of the eCon-devices when the layout has been switched to offline mode. Masking the outputs is done by means of a 16-digit hexadecimal number. Depending on device type the interpretation of the Bit-assignment in the masks differs. With the third selection (free) it is possible to setup an individual mask. Even other values than zero can be set to the outputs. But the latter option is rather unusual because it's generally expected that the outputs are set to zero when the layout is switched to offline mode.

↗ *See also chapter 5.2.9*

Altered settings will only be applied after clicking on the button "Save configuration".

A more detailed description of the system configuration for the use of eCon-devices is available in the special ecOn-documentation:

hw_man_econ_en_A4.pdf

2.5.4. Menu ↪File ↪System settings ↪FOB IO / FOB-M

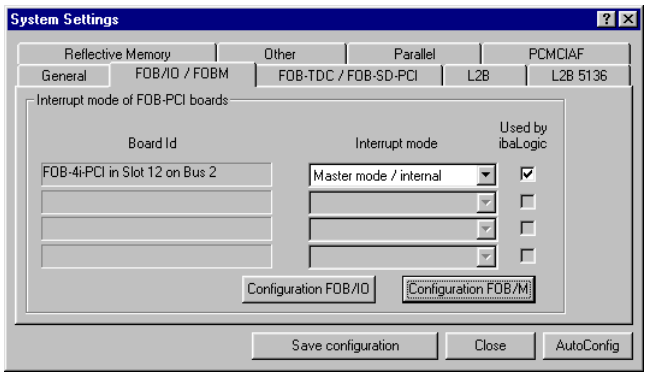


Fig. 28 System settings, FOB IO / FOB-M

❑ ***Interrupt mode of FOB-PCI boards***

- *Board ID*: Display of installed iba PCI cards, auto-detected
- *Interrupt mode:* to be selected; Master mode internal / external or slave mode; only one iba PCI-card must set to "Master mode"!
- *Used by ibaLogic:* yes / no, please check the box if the related card should be used exclusively by ibaLogic (and not by ibaPDA or other programs).

Mouseclick on the "Configuration"-buttons opens the dialog windows which can also be reached via menu ↪*File* ↪*PCI Configuration*, see also chapters 2.6.1 and2.6.2.

Altered settings will only be applied after clicking on the button "Save configuration".

Remark:

The checkboxes "*As FOB-M*" in former versions (< 3.88) have been removed. The settings for a fast data acquisition (sample rate 25 kHz) with Padu8 M, Padu8 ICP or Padu16 M and the card runnning in FOB-M mode should be done in the dialog *Configuration FOB/IO* (FOB-F PCI settings). Each processor of a FOB 4i PCI-card can be set to FOB-M mode individually. Thus a mixed operation of FOB-F and FOB-M mode is possible.

2.5.5. Menu ↪File ↪System settings ↪FOB-TDC / FOB-SD-PCI

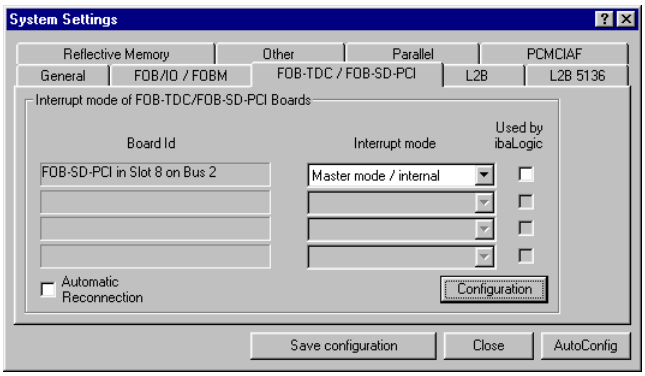


Fig. 29 System settings, FOB-TDC / FOB-SD-PCI

❑ ***Interrupt mode of FOB TDC/FOB SD PCI boards:***

- *Board ID:* Display of installed iba PCI cards of this type, auto-detected
- *Interrupt mode:* to be selected; Master mode internal / external or slave mode; only one iba PCI-card must set to "Master mode"!
- *Used by ibaLogic:* yes / no, please check the box if the related card should be used exclusively by ibaLogic (and not by ibaPDA or other programs).

❑ ***Automatic Reconnection***

If the target system (Simadyn D / Simatic TDC) has been shut-off during operation or is not available due to other reasons the corresponding i/o are blocked because the related drivers are stopped. The i/o are shown as invalid in the layout if the option "unavailable signals are invalid" has been set in the dialog ↪*File* ↪*System settings, General* (see 2.5.1). Other i/o which are not connected to the missing system, e.g. from FOB IO cards, are not affected and will be evaluated.

Selecting this option will urge ibaLogic to establish the communication to the target system and restart the drivers after the target system has returned (which is detected by ibaLogic automatically). This procedure takes approximately 5 to 20 seconds. For this time the evaluation of the layout is completely halted, i.e. no i/o are available. For that reason the selection of this option should be made carefully in order to avoid unwanted effects on the process.

Mouseclick on the "Configuration"-button opens the dialog window which can also be reached via menu ↪*File* ↪*PCI Configuration*, see also chapters 2.6.4.

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.6. Menu ↪File ↪System settings ↪L2B

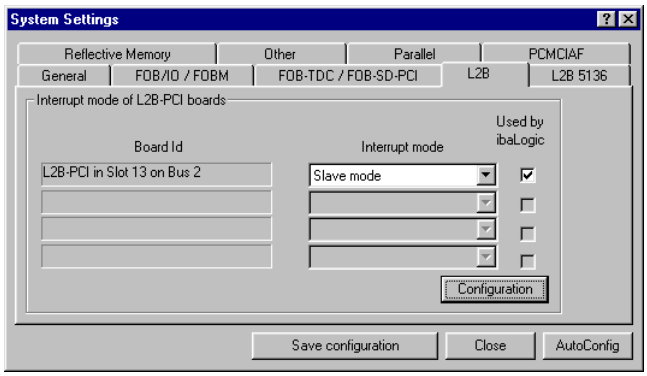


Fig. 30 System settings, L2B

❑ ***Interrupt mode of L2B-PCI boards:***

- *Board ID:* Display of installed iba PCI cards of this type, auto-detected
- *Interrupt mode:* to be selected; Master mode internal / external or slave mode; only one iba PCI-card must set to "Master mode"!
- *Used by ibaLogic:* yes / no, please check the box if the related card should be used exclusively by ibaLogic (and not by ibaPDA or other programs).

Mouseclick on the "Configuration"-button opens the dialog window which can also be reached via menu ↪*File* ↪*PCI Configuration*, see also chapters 2.6.3

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.7. Menu ↪File ↪System settings ↪L2B 5136

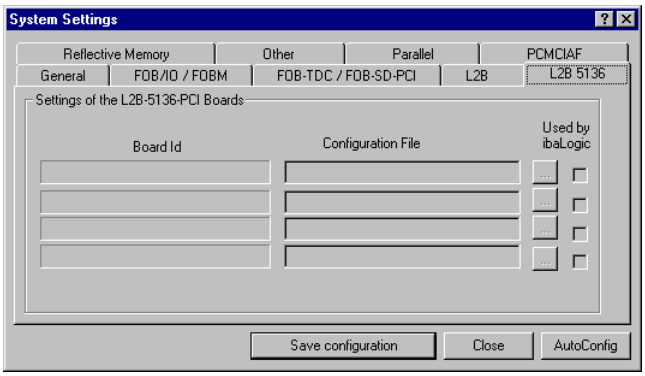


Fig. 31 System settings, L2B 5136

❑ ***Settings for L2B 5136 boards:***

- *Board ID:* Display of installed iba PCI cards of this type, auto-detected
- *Configuration file*: Enter path and file name of the configuration file or browse and select an existing file.
- *Used by ibaLogic:* yes / no, please check the box if the related card should be used exclusively by ibaLogic (and not by ibaPDA or other programs).

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.8. Menu ↪File ↪System settings ↪Reflective Memory

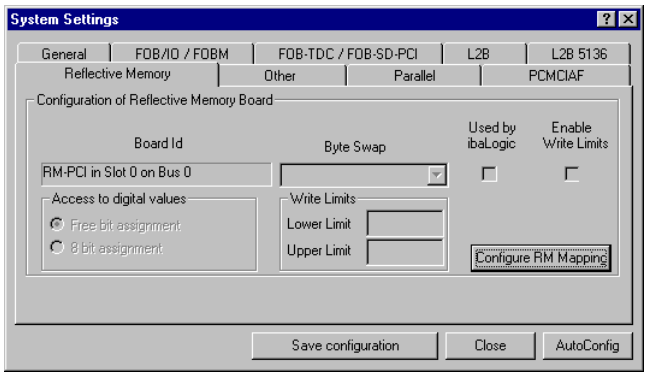


Fig. 32 System settings, Reflective Memory

❑ ***Configuration of the Reflective Memory boards***

- *Board ID:* Display of installed iba PCI cards of this type, auto-detected
- *Byte Swap*: Activate / inactivate the swap mode; depends on the connected system. To be used, e.g. if the target system requires Big Endian mode. Choices: *No Swap, Byte Swap, Word Swap, Byte and Word Swap* and *Swap on Size*.
  Remark: The new RM-board VMI5565 does not support the swap mode any more. The boards VMI5576, VMI5579 and VMI5586 still support the swap mode.
- *Used by ibaLogic:* yes / no, please check the box if the related card should be used exclusively by ibaLogic (and not by ibaPDA or other programs).
- *Activate Writing Limits*: Check the box if the writing limits should apply.
- *Access to digital values*: Select whether the access to digital values should be performed bitwise or bytewise.
- *writing limits*: Preset of the lower and upper writing limits; entry is only allowed when "*Activate writing limits*" is checked.

Mouseclick on the "Configuration RM"-button opens the dialog window which can also be reached via menu ↪*File* ↪*PCI Configuration*, see also chapters 2.6.5

Altered settings will only be applied after clicking on the button "Save configuration".

2.5.9. Menu ↪File ↪System settings ↪PCMCIAF

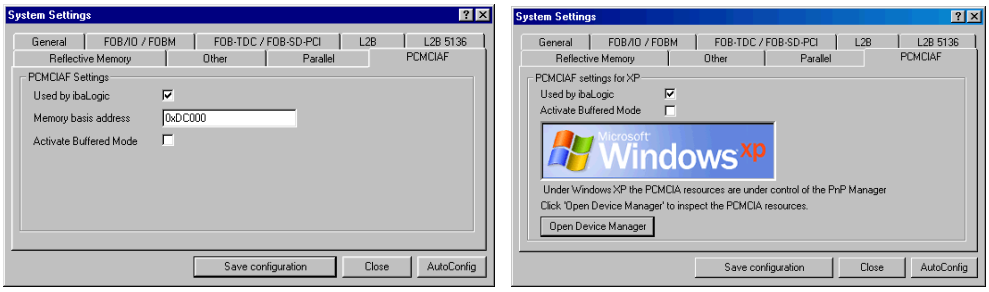


Fig. 33 System settings, PCMCIAF on Windows NT (left) and XP (right)

❑ *PCMCIAF Setup*

By means of the PCMCIA-support ibaLogic can be supplied with input signals even when running on a notebook computer. The card PCMCIAF from iba (order no. 1.020) should be used for this purpose. If the PCMCIAF card should be used please check the box *Used by ibaLogic*. The incoming signals (max. 64) will be assigned to the first two modules of the FOB-F input resources for analog and digital values.

The *basic memory address* is automatically set. It may be adjusted during installation of the card.

The checkbox *Activate Buffered Mode* should be checked, if sampling rates of incoming signals are higher than the task cycle time of the layout in ibaLogic. In this case the input resources FOB-F Buffered Mode should be used in the layout (see 5.1.2).

With Windows XP the card management is provided by the device manager in a more convenient way than with NT.

Altered settings will only be applied after clicking on the button "Save configuration".

2.6 PCI configuration

The menu ↪*File* ↪*PCI configuration* provides access to the same configuration dialogs for selected cards like the "Configuration.."-buttons in the dialogs of the system settings (compare chapters 2.5.2 to 2.5.8).

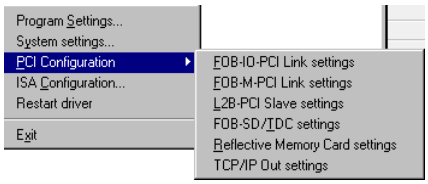


Fig. 34 Menu PCI Configuration

2.6.1. FOB-IO-PCI Link settings

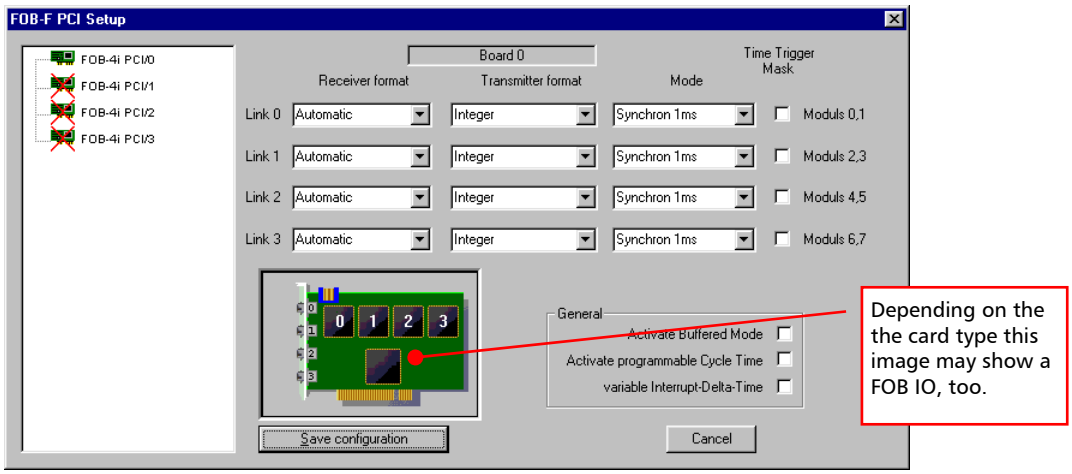


Fig. 35 FOB-M-PCI Link settings

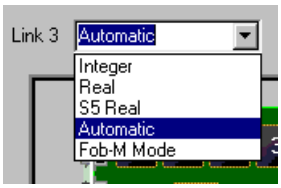
This dialog shows the configuration of each fiber optical link (0...3) of up to four FOB-F-cards (e.g. FOB IO or FOB-4i-PCI). After selection by mouseclick in the tree (left) the configuration can be changed for installed and selected cards. If Fob-M mode is required due to high sampling rates the Fob-M mode can be activated on a per-link basis in the selection of receiver and transmitter format.

FOB-4i-X cards and FOB-4o-X cards work only in F-Mode and M-Mode. The X-Mode (32 Mbit Telegram) couldn't be used in ibaLogic-V3.

FOB-2i-X cards and FOB-2io-X cards are displayed as FOB-4i cards, but only two links are available.

Receiver format

Data format of incoming signals (via optical link); recommended setting: Automatic (default);



Integer: for data coming from SM64, SM128V, ibaNet750 and Padus
Real: for data coming from SM64, SM128V
S5 Real: for data coming from SM64 in S5 Real-format; the SM64-card must operate in the same mode.
Fob-M Mode: for data coming from Padu8 M, Padu8 ICP or Padu16 M with fast sample rates up to 25 kHz. When choosing FOB-M mode, the same format for receiver and transmitter is enforced. Each processor of a FOB 4i PCI-card (=link) can be set to FOB-M mode individually. Thus a mixed operation of FOB-F and FOB-M mode is possible.

❑ ***Transmitter format***

...as above but for sending data

❑ ***Mode***

- *Synchron 1 ms*: The data are received synchronously to the internal basic samplingtime (1 ms) from the connected peripheral components. This is the usual mode for reception of incoming data from FOB-F, FOB IO und FOB 4i PCI cards.
- *Asynchron 1...10 ms*: The data are received with a different sample rate than the basic samplingtime.

↗ *See also* Characteristics of the asynchronous mode, *S. 2-42*

❑ ***Time Trigger Mask***

Release for using programmable sample rates with the related fiber optical port. This is a precondition for operating in asynchronous mode and thus must be checked.

Nearby the checkboxes for the time trigger mask, you'll find the corresponding module numbers as a remark. Each fiber optical link corresponds to two modules, consisting of 32 analog and 32 digital signals each, i.e. a total of 64 analog and 32 digital signals per link.

❑ ***General***

- *Activate Buffered Mode*: If checked, the received data are buffered and then provided to the ibaLogic layout as amn Array-resource (max. buffer depth = 256). This feature is only available for the first eight FOB-F-modules (compare chapters 5.1.2 and 5.2.2).
- *Activate programmable Cycle Time*: If checked, it is allowed to set the sampletime for a fiber optical port of therelated card in the layout.
- *variable Interrupt-Delta-Time*: If checked, the time-lapse between two interrupts may vary.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

2

2.6.1.1. Characteristics of the asynchronous mode

The intention of using the asynchronous mode is to adjust the sampletime of the Padus to the measuring scenario, e.g. for a FFT with as less samples as possible.

The following preconditions are required:

1. The ralated fiber optical port is set to asynchronous mode.
2. Option "*Activate programmable Cycle Time*" is checked.
3. Option "*Variable Interrupt-Delta Time*" ist active (checkmark).
4. The FOB-F-card, whose first Link is running in asynchronous mode must generate the interrupt for ibaLogic.
5. The interrupt setting for the FOB-F-card is "Master/External".
6. The fiber optical link is a closed loop (input/output of the card connected to output/input of the signal source).
7. ibaLogic runs in signal manager mode (↪File ↪System settings).

If these preconditions have been carried out, the following statement applies:

The physical unit]ms] for samplingtime and task-evaluation interval will be replaced by the number of interrupts. As a consequence the time-lapse between two interrupts and thus between two evaluation intervals is variable.

The value *EvalDeltaTime* which is given to the task in ibaLogic is adjusted and corresponds to the real lapsed time.

2.6.2. FOB-M-PCI Link settings

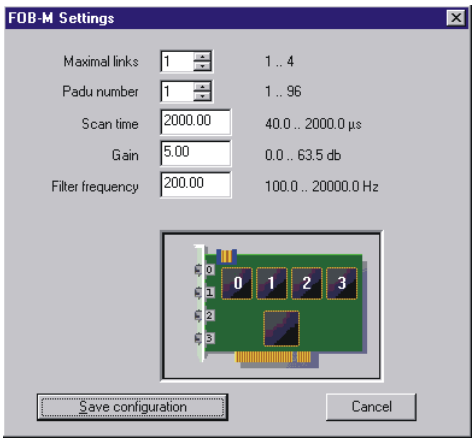


Fig. 36 FOB-M-PCI Link settings

This dialog can be used for setting the default values of the card for operating in FOB-M mode.

These presets apply generally to all links which are set to FOB-M mode.

Usually, the parameters are adjusted individually for each processor (link) later in the layout. The settings made in the layout overwrite the default settings.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

2

2.6.3. L2B-PCI Slave settings

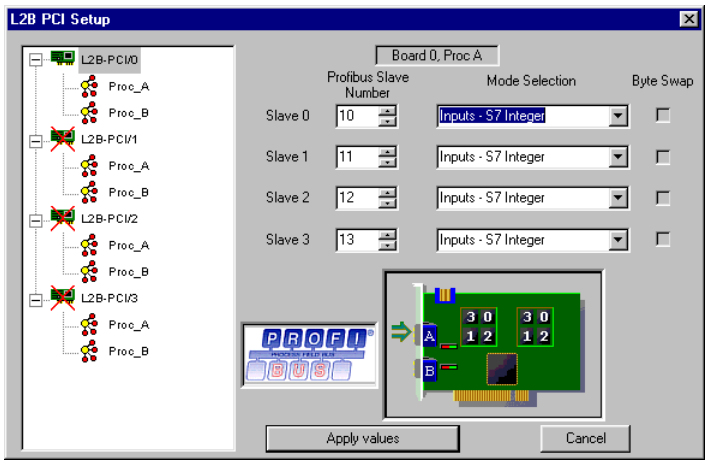


Fig. 37 L2B-PCI Slave settings

This dialog shows the configuration of up to four L2B-PCI cards, each with two processors with up to four Profibus-slaves. After selection by mouseclick in the tree (left) the configuration can be changed for installed and selected cards / processors.

Not all modes are available with all firmware versions. Also, for older ibaLogic versions some functions are not available.

- ❑ The default setting of the Profibus-slave numbers should be adjusted with reference to the Profibus configuration (engineering).
- ❑ The mode for data processing respectively for data type may be set individually for each slave. The modes "flatness..." are dedicated to data which are supplied by Siemens flatness measurement systems. (compare chapter 5.1.5). Moreover, each slave can be deactivated individually, if it's not needed.
- ❑ The selection of the byte swap option (checkbox) depends on the connected target system.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

2.6.4. FOB-SD / TDC Link settings

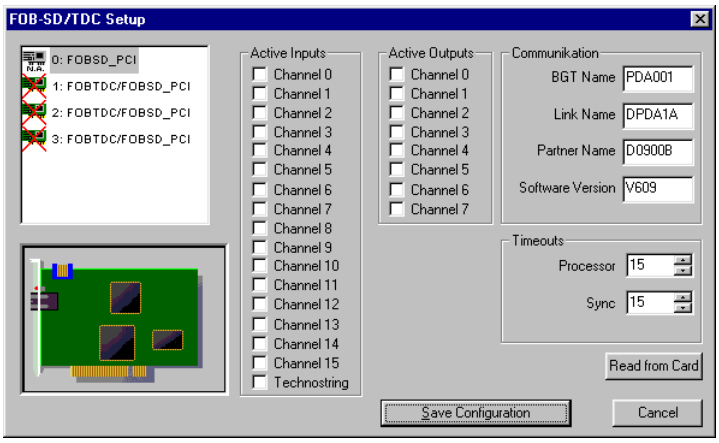


Fig. 38 FOB-SD / TDC link settings

This dialog shows the configuration of a FOB-SD or FOB TDC card. The configuration settings of the card may be changed.

❑ ***Active Inputs***

One or more out of 16 input channels supposed to be used for data transfer can be activated by checkmarks in the boxes (0...15). One channel x corresponds to one FOB SD/TDC – Simadyn Lite module x in the resource area of ibaLogic (analog + digital, x = 0..15).

Please note, that for each selected input channel a transmission telegram MxPDADAT (x = 0 .. 9,A .. F) must be provided in Simadyn D, resp. in Simatic TDC.

❑ ***Active Outputs***

One or more out of eight input channels supposed to be used for data transfer can be activated by checkmarks in the boxes (0...7). One channel x corresponds to one FOB SD/TDC – Simadyn Lite module x in the resource area of ibaLogic (analog + digital, x = 0..7).

Please note, that for each selected output channel a reception telegram PDAMxDAT (x = 0...7) must be provided in Simadyn D, resp. in Simatic TDC.

Because the communication principle of ibaLogic in this case is very similar to the communication between Simadyn D and ibaPDA, you'll find further information in the design guidelines for ibaPDA, ***sw_man_ibaPDA-SD_Project...*** *resp.* ***sw_man_ibaPDA-TDC_Project...,*** *which are available for download on our website.*

❑ ***BGT Name***

This is the local PC-system name. In terms of SD or TDC it correponds to the name of the SD /TDC subrack. Changing the default setting "PDA001" is not necessary.

❑ ***Link Name***

This entry is the unique name of the connection between the FOB SD/TDC-card and the target interface (CS14 board / GDM). This setting has to be changed if the communication partner is connected to othe ibaLogic or ibaPDA systems. If the name is not unique the error message =0x6AA0 will be displayed.

2

❑ ***Partner Name***

This entry is the name of the connected interface board (CS14- or GDM-processor). It must be entered here! You can find the name in the engineering documentation of SD resp. TDC or by using the BGT diagnostics in ibaDiag. If name is not correct the error message =0x6AA6 will be displayed

You'll find further information concerning this topic in our manual about ibaDiag ***sw_man_ibaDiag_en_A4.pdf****, chapter 2.3.11 (or .._LTR.pdf for letter-format).*

❑ ***Software Version***

This entry shows the version number of the Simadyn D resp. TDC basic software package. It must be entered here! You can find the name in the engineering documentation of SD resp. TDC or by using the BGT diagnostics in ibaDiag. If name is not correct the error message =0x6AB3 will be displayed.

You'll find further information concerning this topic in our manual about ibaDiag ***sw_man_ibaDia g_en_A4.pdf****, chapter 2.3.11 (or .._LTR.pdf for letter-format).*

❑ ***Timeouts***

These entries show the waiting time for the acknowledgements of commands to the FOB-SD/TDC card. Usually, the default setting "15" must not be changed.

❑ ***Button:*** **Read from card**

By pressing this button the data which are required for establishing a logical connection, i.e. BGT-Name, Link Name, Partner Name and Software Version, will be loaded from the Simadyn D resp. TDC system and entered in the corresponding fields, provided the physical connection is ok.

❑ ***Automatic Reconnection***

If this box is checked off the data which are required for establishing a logical connection, i.e. BGT-Name, Link Name, Partner Name and Software Version, will be loaded from the Simadyn D resp. TDC system with every driver restart and entered in the corresponding fields, provided the physical connection is ok.

This option may be used for automatically reestablishing a connection when a link got lost.

The automatic reconnection should be handeled with care!

Since it takes approximately 5 seconds to reestablish a connection there might be an interference with a proper execution of the ibaLogic layout, because during this time the evaluation of the ibaLogic layout is halted.

Make sure, that the process or machinery which is to be controlled by ibaLogic is in save condition when activating this option.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

2.6.5. Reflective Memory Card settings

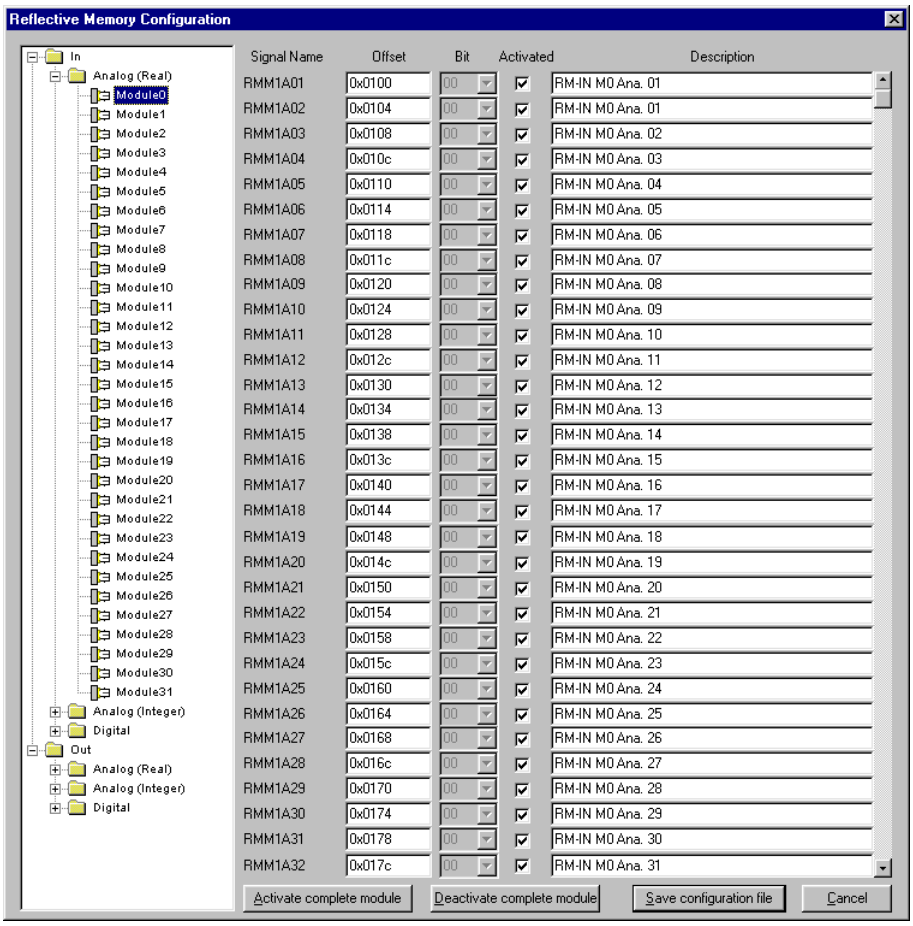


Fig. 39 Reflective Memory card settings

The definition of addresses and symbolic names for input and output variables which are exchanged via Reflective Memory with other systems should be done in this dialog window. The corresponding input and output resources are available in ibaLogic (32 modules with 32 analog values each, REAL or Integer, and 32 modules with 32 digital signals each). The related modules are shown in the left part of the dialog window.

The settings depend considerably on the connected system. The addresses and symbolic names shown as default settings in the dialog window are presets for example and subject to change if necessary.

❑ *Signal Name*

This column shows the signal names which are used internally by ibaLogic and which cannot be changed. These names are also shown in the tooltip when the mouse points on the connection point of the input or output in the layout.

❑ *Offset*

The offset or memory address of each signal in the reflective memory should be entered in this column. The default settings show typical entries for example:

Real signals from 0x0100 in 4-byte-steps with module distance of 0x0100, integer signals from 0x0180 in 2-byte-steps with module distance of 0x0200 and digital signals from 0x0080 in one doubleword (= 32 bit) with module distance 0x0002.

2

In case of a point-to-point connection between ibaLogic and another system, i.e. if the data can be mapped in a memory block, a similar addressing is very likely. But Reflective Memory (RM) allows also the linking of several systems in a ring topology for data exchange which is not related to ibaLogic as well. In such a case there is a free choice of addresses, i.e. the addressing may be adjusted to the RM-configuration.

There is no rigid assignment between RM-address and ibaLogic variable. It is not necessary to arrange the data in the iba module structure.

❑ ***Bit***

These fields are activated only if a module for digital signals is selected. Digital signals should be packed in double words (DWORD, 32 bit) for ibaLogic. A single signal in a double word is addressed by the bit number. The bit addressing may be adjusted to the configuration-related requirements as well.

❑ ***Activated***

These checkboxes may be used in order to inactivate single analog or digital signals of a module if they are not needed in ibaLogic or if the must not be used by ibaLogic.

❑ ***Description***

The description is a simple customized text entry which will be used as signalname in the layout, resp. the function block diagram. The description will appear as signalname in the input / output margins of the layout and in the input / output resource trees as well.

❑ ***Buttons „Activate / Deactivate complete module“***

These buttons activate resp. deactivate all signals of a selected module.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

In case you have to define many signals it may be a painstaking task to enter all signals in this dialog window. There is a way to ease your work:

The RM-settings are stored in an ASCII-file dynconf.cfg in the path …\configuration in the program directory of ibaLogic.

This is a csv-file which may be opened with MS Excel (e.g. rename the file before to .csv). The settings can be processed more efficiently by using the means of MS Excel.

Finally, save the file again under its orinal name.

↗ *See also 5.1.6*

2.6.6. TCP/IP Out settings

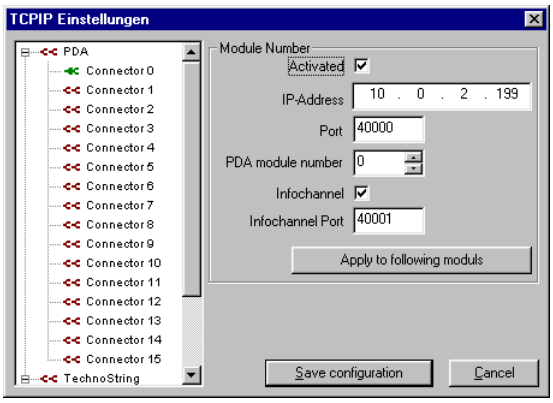


Fig. 40 TCP/IP Out settings

This dialog serves for setting up the usage of output signals via a TCP/IP connection.

In compliance with the TCP/IP OUT output resources (refer to section 5.2.5) up to 16 modules with 32 analog and 32 digital signals each may be sent towards an ibaPDA-system. Furthermore, there are four string variables availble for transmitting Technostring outputs. In order to make use of these output resources the TCP/IP channels have to be configured and activated. 16 channels (connector 0…15) which can be configured and activated individually are provided for 16 modules of output signals towards an ibaPDA-system. Thus, connections of unused modules may be deactivated or connections may be assigned to different target ibaPDA-systems using different IP-addresses.

The data to be transmitted to an ibaPDA-system may be assigned on base of an output module to any module in ibaPDA.

❑ ***Modul number***

- *Aktivated*: The related and selected connection will be activated only if this box is ticked off.
- *IP-Address*: IP address of the receiver; this may be also the local IP address if ibaLogic and ibaPDA are running on the same PC.
- *Port*: Hier muss die gleiche Portnummer eingetragen werden, die im ibaPDA-System in den Systemeinstellungen eingetragen ist.
- *PDA module number*: The number of the module in ibaPDA where the data are supposed to be assigned to should be entered here.
- *Infochannel*: no function; in preparation for transmission of signal names
- *Infochanne Port*: no function
- *Button „Apply to following modules“:* Using this button will copy the settings of the currently selected module to the modules beneath in the list.

Altered settings will only be applied after clicking on the button "Save configuration" or respectively "Apply" + "Save configuration".

For using ibaLogic outputs in ibaPDA via TCP/IP the following conditions apply: a) the option "TCPIP ibaLogic to PDA" must be released in the dongle, b) the port number must be entered in the system settings in ibaPDA, c) the moduletyp "IbaLogic" must be selected in the module configuration in ibaPDA.

3 Working with ibaLogic

ibaLogic provides a variety of functions and there are many ways to find a solution for a problem. Before starting the engineering process it is important that there is a good comprehension of the structure and the philosophy of ibaLogic, which is described in the following.

3.1 System limits and boundary conditions

It was our explicit intention not restrain the capabilities of ibaLogic, concerning number of flags, I/Os etc., by build-in limits like it is done for many other control systems on the market due to technical or marketing reasons. On one hand this freedom is an advantage for the customer, on the other hand it might be mistaken for a "never-ending pot".

On principle, every system has its limits in terms of processing capacity, i.e. only a limited number of operations in a time interval can be processed. In case of an open system like ibaLogic these limits are determined by parameters such as CPU power, memory size or other hardware-dependent factors of the environment for ibaLogic. When creating a control application the knowledge of the interaction between the different factors is important in order to avoid an overload of the system by using its powers in all directions to their full extend.

Basically, a few restrictions apply:

The display "Evaluation [%]" should not exceed 100 %, (i.e. 1.0)!

The bigger the program the more likely are delays in compiling when making online modifications (without HotSwap)!

In the latter case it depends on the kind of modification. If a modification affects only one task it may work without noticeable delay (tens of ms). But if more tasks are concerned, e.g. when modifying an OPC, it may occur that the entire project (layout) must be compiled, linked an located. If a layout contains around 350 pages, this operation may take one or two seconds (on a double Pentium 3 with 1 GHz, plenty of RAM and "Eval %" almost 100%). And this could cause a real bad behaviour of the controlled machinery!

Due to extended compilation time and depending on system load and kind of modification the online process may be affected and halted for some seconds when online modifications are performed. The system outputs won't be refreshed in this time!

In this case there may be hazard for life or machinery!

We recommend to use the Hot-Swap method when changing the layout during operation.

Always secure the layout against unauthorized or unintended modification by using the password protection and lock function of ibaLogic.

Depending on the layout size the creation of a Hot-Swap layer might take some tens of seconds (in the above mentioned example around 20 - 30 s for each switch-over). But the safety benefit is worth it.

3.2 Important terms and functions

The functionality of the application is described by functions, function blocks, macros, connection lines and comments in ibaLogic. The container of all the tasks is called "project".

It starts with the creation of a new project. The project contains an application-dependent number of tasks which run with on a particular time base each (cycle time as a multiple of the basic ibaLogic samplingtime, respectively the FOB-board samplingtime). The contents of a project is to be stored as a file with *.lyt* extension. A project is always additionally stored as a "Structured Text"-ASCII formatted file according to IEC1131-3.

One of the most innovativ features of ibaLogic is the capability to switch over to offline evaluation mode immediately without waiting when working on the graphical programming for test and diagnostics purposes. Thus, the function of a program or the behaviour of function blocks may be tested quick and easy.

In order to switch on the evaluation mode click the  button in the tool bar.

When testing complex interlockings the evaluation of a single step (cycle) or of a specified number of steps is possible (single step / multiple step). In evaluation mode the outputs are not active but inputs are read.

The activation of a project (layout) and the output of variables to a connected process are done in online mode. In order to switch over in the online mode click the button "*Activate / deactivate Online Evaluation* "  in the tool bar. The background color of the screen switches from grey to purple when working in online mode.

When switching over from evaluation mode to online mode all the outputs are activated according to the engineered application. This may cause unintended reactions of a connected machinery.

Make sure to avoid danger to life due to sudden moves of a machinery or other related effects!

Furthermore, we recommend to perform only little, easy-to-handle modifications in closed-loop controls because the cyclic processing may be affected as well.

In order to prevent unintended reactions of the process it is strongly recommended to use a Hot-Swap layer for working. With a Hot-Swap layer it's possible to make a copy of a task running in online mode, make the changes in the copy and switch back to normal operation afterwards, by applying the changes.

To create a Hot-Swap layer please follow these steps:

1. Switch to online mode by clicking "*Activate Online Evaluation*" 
2. Lock the current online layer by clicking "*Lock Online Layer*" (key) 
3. Then create a Hot-Swap layer by clicking "*Create hot swap layer*"  . This command causes the system to create a copy of the contents of the online layer without quitting the online mode. This copied hot-swap layer may now be modified and testet in evaluation mode without affecting the process. While working on the hot-swap layer the original online layer is executed in the background with highest priority.

4 Switch-over to the modified project by clicking menu ↪ *Hot Swap* ↪ *Apply to Online Layer*
The modified hot-swap layer will be switched immediately to online mode during operation (without loss of control cycles).

Although ibaLogic is capable of switching over smoothly and without loss of control cycles there is always a risk of hot-swap switching due to engineering errors in the application program or wrong parameters. It is always recommended to switch over when the process or machinery is in safe condition.

3.3 Which tasks should run how fast – and what does it mean?

The essential decision in a project is the one about the project structure. Usually, a project is devided into separate tasks which could be completely independant from each other or which could differ from each other in terms of dynamic behaviour. It's clear, that a roomtemperature control can work with a cycle time of 1 s when the cycle time of a hydraulic gauge control in a cold rolling mill must not exceed some tens of milliseconds.

Thus, time is an essential parameter to be considered when dividing a project into different tasks. The shortest cycle time in ibaLogic is 1 ms.

3.4 Relation between task cycle, processing time and evaluation%

iba guarantees that the tasks can be started in intervals of 1 ms but some conditions apply.

According to the definition a task in ibalogic (Version 3.xx) is uninterruptable. This has an impact on the cycle time.

Example: Two tasks are defined. Task0 with 5 ms cycle time and Task1 with 100 ms. For the evaluation Task0 needs 2 ms and Task1 needs 8 ms. These values for the evaluation time can be ascertained with the evaluation statistics. Task1 – which is uninterruptable – runs longer than the cycle time for Task0 (5 ms) requires. In this case the display of Evaluation [%] in the bottom bar of the ibaLogic screen shows a value over 100 % because it shows the relation between the longest evaluation time and the shortest cycle time. For the evaluation of the function blocks, this is not a problem because the function blocks are designed time-relativ. Time-relativ means, that each function block checks how much time has lapsed since it was started (keep that in mind when creating your own function blocks with time-depending elements).

Of course, the obstruction of the (shorter) task could cause some problems, such as missing an impuls with a length of 5 ms which is created by switching-on in one and switching-off in the next cycle. But in order to avoid such problems it is recommended to use special pulse-generating functions, like with Padu8 O.

Having realized these facts it will lead to the following rule of thumb:

The evaluation [%] should never exceed 100%, else obstructions or other side effects are inevitable!

In the ideal case (not a must-be) the total of the evaluation times of all tasks should be less than the shortest cycle time.

Else, obstructions might occur due to interference of tasks and evaluation times.

The current evaluation times of the tasks can be ascertained by use of the evaluation statistic under menu ↪ *View* ↪ *Evaluation Statistic.*

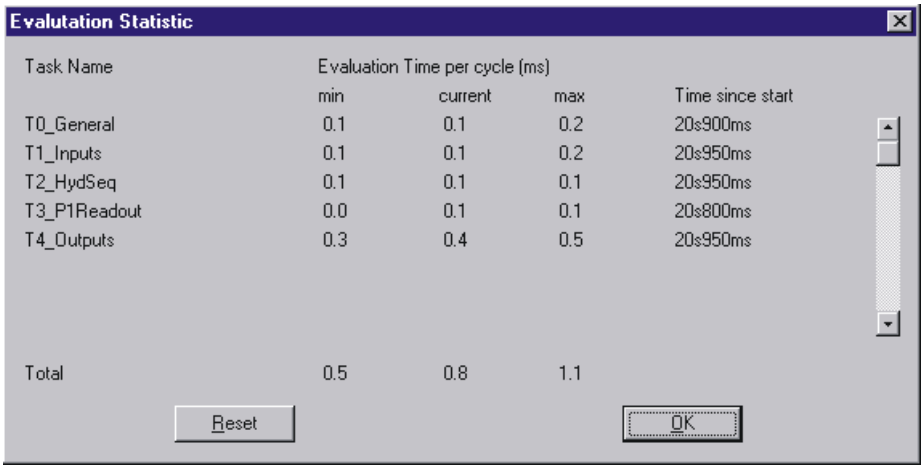
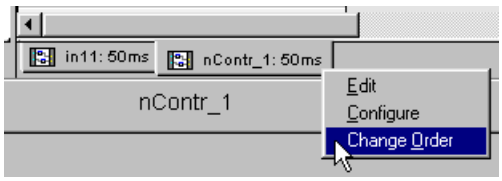


Fig. 41 Evaluation statistic

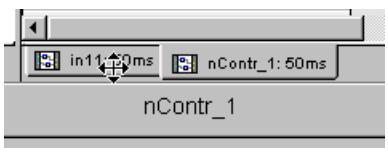
3.4.1. Order of task processing

Due to certain conditions it might be necessary that the order of task processing should be changed. Usually the tasks are processed in the order from left to right. To change the order follow these steps:

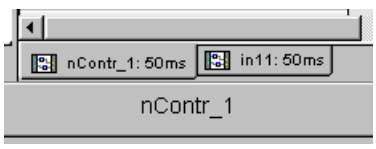
1 Right mouseclick on tab of the task which should be shifted (e.g. task "nContr_1"). Then mouseclick on "Change Order" in the pop-up menu.



2 Then left mouseclick on the tab at the target position for the task, e.g. "in11"; cursor shape altered.



3 Klick left mousekey again. Task "nContr_1" is now on the left side of Task "in11" and will be processed before.



3.5 The I/O system of ibaLogic

Generally, the iba I/O system receives the data independently from the PC-processing. (Of course, an application is required for outputs). This happens usually with a scan rate of 1 ms, i.e. signals will be transferred even if no PC-application is running.

Exception: When using connections to devices which need bidirectional communication, such as Padu8 M, a running ibaLogic application is required.

The following table gives an overview of the I/O components of ibaLogic and their related PC-connection boards.

Peripheral device	PC-connection board	Inputs (I) and/or outputs (O)
Padu8, -16, -32	FOB 4i PCI, FOB IO	I
Padu8 M , -ICP	FOB 4i + FOB 4o PCI, FOB IO	I (outputs for configuration only)
Padu8 O	FOB 4o PCI, FOB IO	O
ibaNet 750	FOB 4i + FOB 4o PCI, FOB IO	I / O
SM 64 IO	FOB 4i + FOB 4o PCI, FOB IO	I / O
SM 128 V	FOB 4i + FOB 4o PCI, FOB IO	I / O
CS12/14/16 (Simadyn D)	FOB SD PCI	I / O
SM64-SD16 Simadyn D (16 Bit)	FOB 4i + FOB 4o PCI, FOB IO	I / O
Simatic TDC	FOB TDC PCI	I / O
Simatic S5, MMC	FOB 4i + FOB 4o PCI, FOB IO	I / O
Simatic S7, Profibus	L2B x/8 PCI, DPM64+FOB	I / O

Table 4 I/O components

3.5.1. Identification and naming of I/O resources

There are several ways to describe resources and I/O signals in ibaLogic. Generally, the name of a signal consists of up to 32 ASCII characters, including special characters and blanks.

- ❑ The resources can be renamed in the tree structur in the left part of the screen (resource area) by two clicks on the signal name or in the resources margins in the program area after it was placed there by doubleclick on its name. If a resource has been renamed, the new name will appear everywhere the resource is used in the program.

- ❑ More than one resources can be exported as a group in a CSV-file. Right mouseclick on a resource in the group, choose *Export*, click OK on question e.g. *Export description for resource tree Analog (Real)?*, give a filename and store. The CSV-file can be edited with an usual ASCII editor or other software, e.g. MS Excel. If the modified CSV-file has been restored, it can be imported by ibaLogic, using the menu ↪ *View* ↪ *Load resource descriptions...* Either signals and signal groups (module names) can be renamed. This function is very helpful if many signals should be renamed.

Please notice that the edited file is stored in the same directory as the source file, particularly when working with MS Excel. The default-directory for the CSV-files in ibaLogic is ...\ibaLogic\configuration.

- ❑ By using menu ↪ *View* ↪ *Equalize resource descriptions* the resource names can be transferred from the project to the resource tree or v.v. This function is useful if project parts of different engineers have to be merged together or if standard projects have to be adjusted to different I/O systems.
  Remark: The link is always the internal variable name in ibaLogic.

- ❑ An I/O-signal which has been placed in the project, i.e. in the function block diagram can be renamed individually by a doubleclick on the signal. As a consequence, one I/O-signal may have different names in different tasks!

All individual name modifications will be reset if an equalization from tree to project is performed again.

3.6 Modes of operation of ibaLogic

ibaLogic offers a variety of operating modes in order to match the needs of different applications. Because ibaLogic may be used as a soft-PLC but as a signal manager, a signal processor or a simulation tool as well, there are several modes of operation.

3.6.1. Signal Manager

The Signal Manager Mode ensures that ibaLogic won't miss any incoming sample even if single tasks have been obstructed, i.e. "Evaluation [%]:" has been > 100 %. The sequence control system of ibaLogic ensures that the data are available equadistant in the selected sampling cycle. In case of task obstruction cyles are even made up for the lost time. In the worst case it could occur that ibaLogic evaluates only "old" values. But it's always ensured that e.g. a FFT analysis can rely on equadistant and correct values.

Output values will be written by each task at the end of its cycle if output resources are connected in the function block diagram.

3.6.2. Soft-PLC

The Soft-PLC Mode which is suited for control and regulation tasks ensures that only the freshest signal values are processed. Unlike in the signal manager mode it doesn't matter whether samples get lost or not. On the contrary, it is intended to process only the freshest data, i.e. data from the last I/O transfer cycle.

The first task of a new cycle samples the input resources. The "aging" of the resources is determined by the basic sampling cycle time which was set in the hardware settings. If this sampling cycle time is set to e.g. 10 ms and the first task has a cycle time of 50 ms, the first task can always process input data which are not older than 10 ms. But they may be younger.

Output values will be written by each task at the end of its cycle if output resources are connected in the function block diagram.

3.6.3. Turbo Mode

The Turbo mode should be activated when using a PC with double-processor. The performance and the reliability can be improved in this case because one processor works only on the application program (runtime) whilst the other cares about administrative tasks related to the operating system (Windows). Particularly when working in soft-PLC mode on control and regulation this option is highly recommended.

3.6.4. Playback

The playback mode is a very useful feature for the simulation of processes.

In playback mode a data file which had been recorded with an iba online aquisition program such as ibaPDA, ibaQDR or ibaScope, may be replayed like a tape recording and thus be used as a source of input signals. The special quality is the fact that real data of a plant or a process are used for simulation and testing, reaching a higher physical fidelity than by process modeling. Especially for revamp projects this is an interesting point.

3.6.4.1. Using the playback function

1 The precondition for using the playback function is the activation (checkbox) of the "Playback mode" in the menu ↪*File* ↪*System settings* ↪*General* (refer to chapter 2.5.1)

2 Furthermore one should decide whether to use hardware I/Os or not together with the playback operation (...↪*System settings* ↪*Other*, refer to chapter 2.5.2)

3 For the configuration of the playback function use the menu ↪*File* ↪*Program settings* ↪*Playback* (refer to chapter 2.4.4). If a valid data file is available in the specified folder, the essential data like starttime, sampletime and number of frames will be displayed in the dialog window.

4 If a certain time range in the data file isn't of interest yet, disable the manual entries of start- and endtime. Select replay mode and repeat mode.

5 Now it's time for the module assignment. The recorded signals which are identified by module- and channel-IDs should be assigned to the input resources of ibaLogic.

3.6.4.2. Module assignment for playback

A mouseclick on the button "Module assignment >>" in the playback dialog window opens the following dialog:

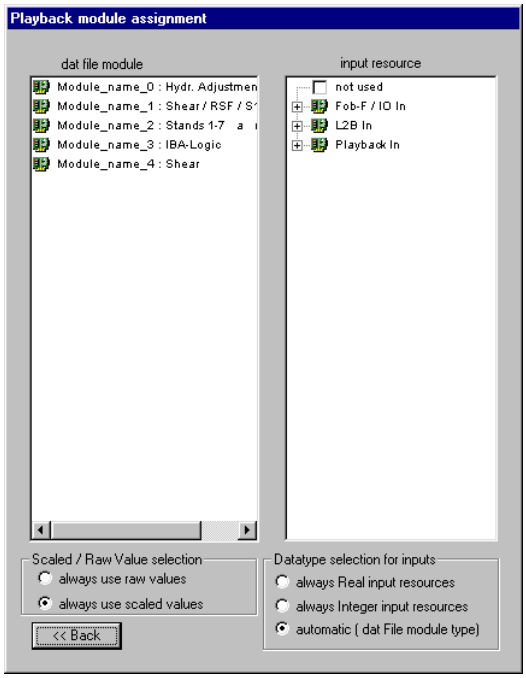


Fig. 42 Playback module assignment

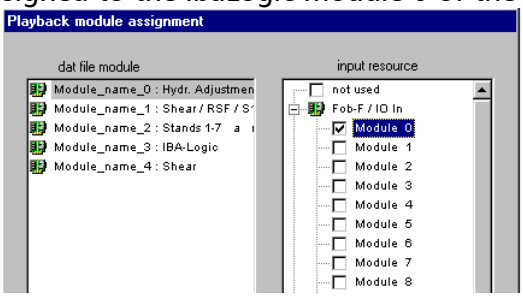
The left part of the window shows the modules as they are stored in the data file and as they had been defined in the acquisition system respectively. The module names are displayed but the names of the signals can not been seen.

The right part of the window shows the input resources which may be used for playback operation. These are resources of the types FOB-F / FOB IO In, L2B In and Playback only.

The assignment concerns modules only (32 analog + 32 digital signals each) and no single signals:

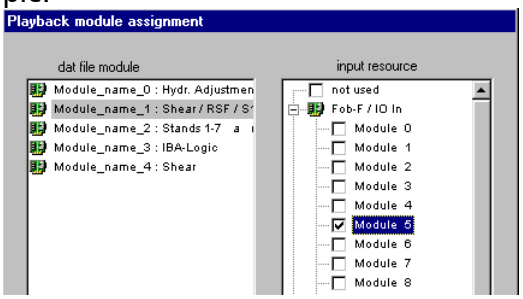
1 First select a module of the datafile in the left field by mouseclick.

2 Then open the tree in the right field for the ibaLogic resources you want to use for playback by clicking on the little "+" and check the module you want to assign to the selected data file module.
Example: All the signals of the data file module no. 0 should be assigned to the ibaLogic module 0 of the FOB-F input resources.



The assignment of data file modules to FOB-F or L2B input modules should be done only if the playback operation "without HW I/O" is selected in the system settings (menu ↪File ↪System settings ↪Other, Playback settings), otherwise the hardware input signals might be overwritten by the datafile signals. If either hardware input signals and data file signals should be used in playback operation simultaneously (mixed operation) it is recommended to assign the data file modules to the PlaybackIn modules.

3 The numbers of assigned modules must not be equal. It is also possible to assign a data file module 1 to an ibaLogic input module 5, for example.



4 After completion of the module assignment the kind of values and the datatype of the inputs may be selected.

- *always use raw values*: ibaLogic takes the signal values as they are stored in the data file. This option will prevent another scaling in ibaLogic of the signals which had been already scaled in ibaPDA or are available in physical units.
- *always use scaled values*: ibaLogic takes the signal values from the data file and scales them using the "*minscale*" and "*maxscale*" information which is stored in the data file with each signal as well.
- *always Real input resources*: All analog input resources will be evaluated as of datatype REAL.
- *always Integer input resources*: All analog input resources will be evaluated as of datatype INTEGER.
- *automatic*: The input resources will be evaluated according to the datatype stored in the data file.

These five settings may be used in combination, but just a few make sense:

Datatype in data file	always raw values	always scaled values	always Real	always Integer	always automatic
INT16		●	●		
INT16		●			●
INT16	●			●	
REAL	●		●		

● = combinations that make sense

The playback operation will be started finally by activating the evaluation mode, or the online mode, respectively.

3.7 Fault management

3.7.1. Zeros on broken links

The activation of this option causes a reset of all input signal values of a module to zero (0) in case of a communication breakdown between an FOB-F / FOB 4i board and the peripheral devices. The advantage is to set a defined and safe state of the input side in case of a malfunction. If this option is not active in case of a fault the latter input values will remain.

3.7.2. Unavailable signals are invalid

Signals are unavailable when the related PC-board which the input signals are assigned to is not there or not working. If this option is selected, the unavailable signals will be marked as "invalid" in the ibaLogic layout (see below).

If a PC-board is installed and working, then the assigned signals are considered as available.

Disconnecting the fiber optical cable or switching off a peripheral device, e.g. a Padu, will not cause the system to declare the signals as "unavailable"!

Signals will be marked as "invalid" in the layout by a red frame. Because the status "invalid" of a signal or variable can be passed on, also the variables which derive from computations or interlockings with invalid variables will be marked as invalid too.

3.8 ibaLogic handling

3.8.1. Drag & drop

The handling of ibaLogic is done usually by simple drag & drop methods like in many other Windows NT® applications. I/O-signals or function blocks in the resource area can be selected by a left mouseclick (hold) and "dragged" into the required area, e.g. input signal margin, program area or output signal margin.

3.8.2. Right mousebutton

Using the right mousebutton anywhere in the program area or in the input/output signal margins will open a window with the "*Edit*"-menu functions as described in chapter 2.3.2.

Using the right mouse button on an input or output signal in the resource area will offer opportunity for resource group export as described in chapter 3.5.1.

Using the right mouse button on a tab in the task selection bar will open a menu for task settings as desribed in chapter 3.4.1. and 3.8.3.

3.8.3. Adjust the size of the program area of a task

The initial size of a task's program area is one page. If this is not enough space for an application, the size can be adjusted individually for each task. There are two ways to change the size or to add more pages, respectively:

1. Place the cursor on the lowest or on the far right borderline (cursor shape switches to ↕, resp. to ↔), press the left mouse button and drag the border slightly down, resp. to the right and a new page will be added below, resp. on the right side.

2. Another way is to use the menu ↪*Edit* ↪*Task* ↪*Configure Task...*, in order to open the window "*Task Settings*" where the number of pages in horizontal and vertical direction can be adjusted, in the example below a total of 10 pages.

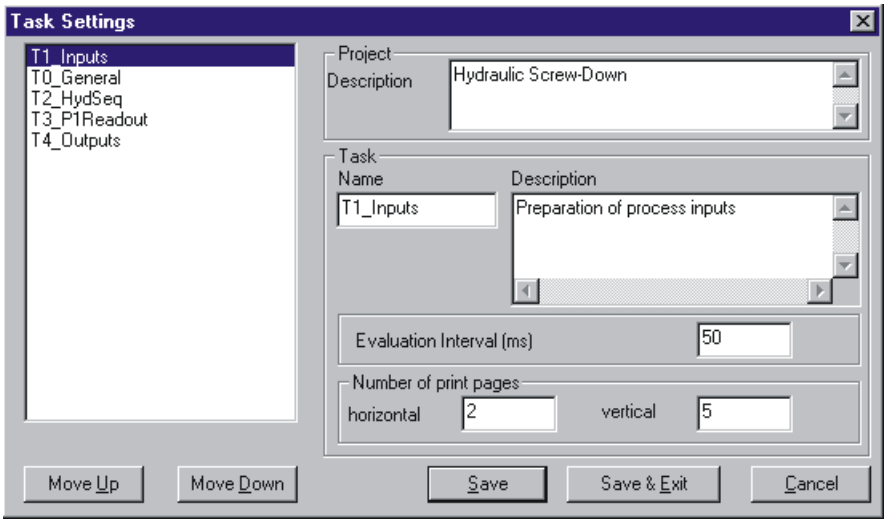


Fig. 43 Task settings dialog

3.9 Selection and connection of function blocks

The engineering of the application is done by use of function blocks. By clicking on the "*Functions*"-tab at the bottom in the resource area one switches from the resources to the function block directory. For the purpose of a better clearity, the function blocks are subdevided in seven groups.

- ❑ Basic Functions
- ❑ Basic FBs
- ❑ Global Variables
- ❑ Global FBs and Macros
- ❑ Global DLLs
- ❑ Local FBs and Macros
- ❑ Local DLLs

The function blocks are described in detail in 4".

After selection of the desired function block, e.g. the multiplier "*mul*", from the directory "*Basic Functions ↪arithmetic*" by use of the left mouse button, just drag it into the program area and let it drop.

All other function blocks can be placed in the program area in that way.

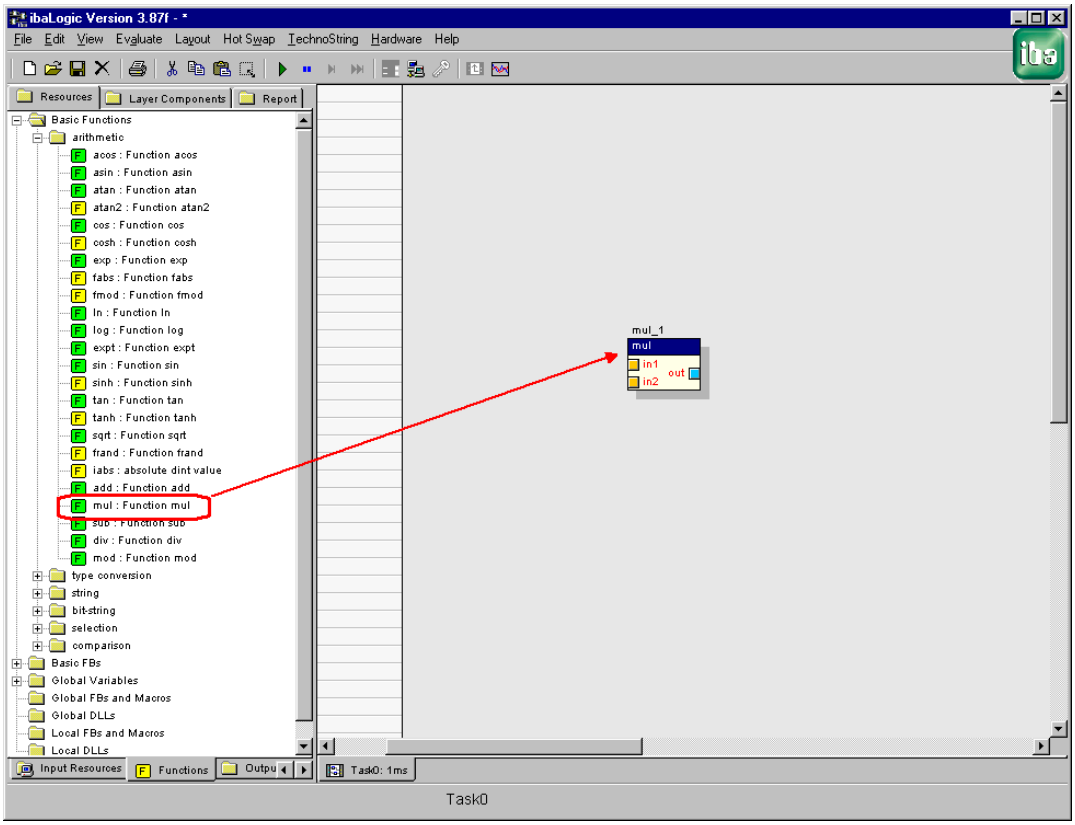


Fig. 44 Placing a function block in the layout

3.9.1. Connection lines and branching

ibaLogic provides three types of connections: connection lines, IntraPage connectors and OffTask connectors.

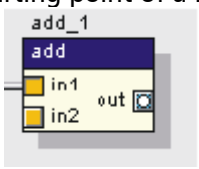
In order to connect one function block with another, just click on the in- or output of the first function block and drag the line to the out- or input of the other function block.

There are three types of lines which are classified as belonging to different data types and which are represented in different colors.

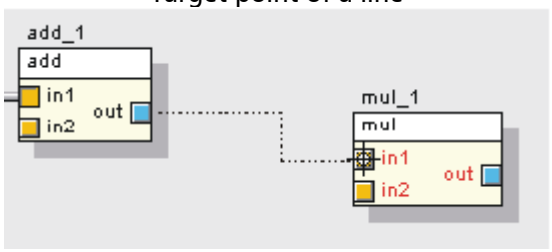
- Binary connections; they show the current logical state of a line, i.e. of the represented signal:
  blue = low / FALSE, red = high / TRUE (in online or evaluation mode)
- All other datatypes are represented by grey connection lines, i.e. INT, REAL, LREAL etc.
- Arrays, resp. vectors, are represented by green lines. Only arrays of the same lentgh and datatype can be connected with each other. If the size of an array changes, the connection has to be cut first and reinstalled after.

The *drawing* of lines is done easily by placing the cursor on the sensitive area of a function block or an I/O-resource (cursor shape changes to ⌑), press the left mouse button (hold), drag the cursor over the target connection point and let the mouse button go. (If a valid connection point is recognized, the cursor shape switches to "cross-hair sight"). The routing of the line is done automatically.

Starting point of a line

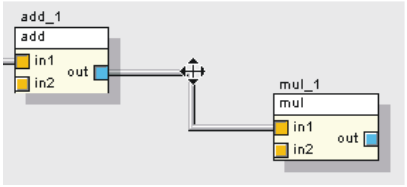


Target point of a line

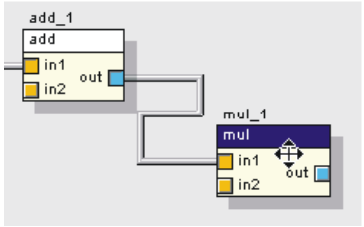


If the *route* of a line shall be changed, this can be done by placing the cursor on a kink of the line (cursor shape changes to an 4-arrow-cross), pressing the left mouse button (hold) and drag the line to the new position. If the objects to be connected are too close to each other the auto-router may create loops or meandering lines. To avoid this, move the blocks more apart.

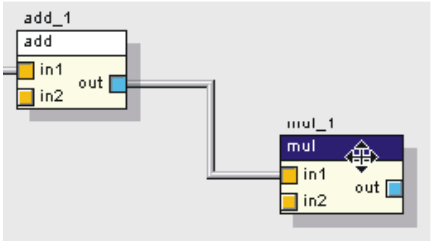
Change line route



Line routing, objects too close

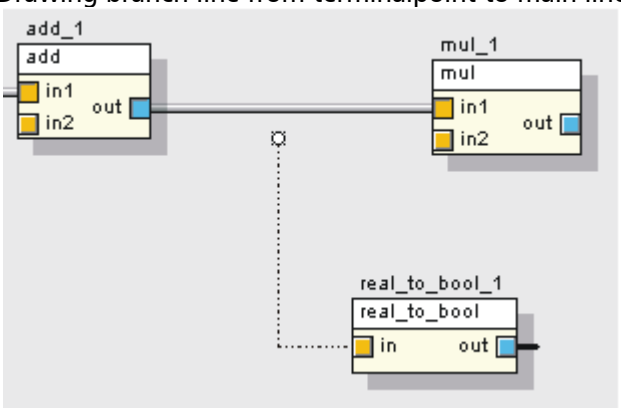


Better line rout, objects apart

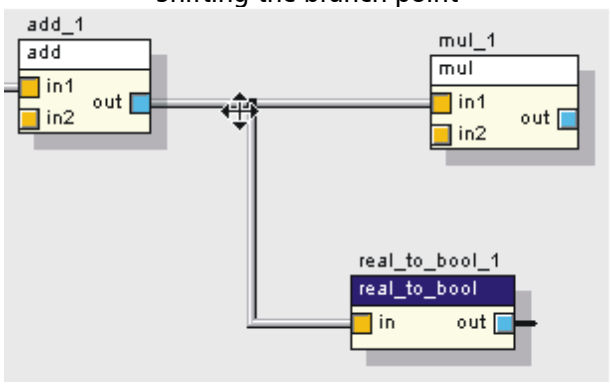


Line branches are created by drawing backwards from the target point of the new line to a point on the main line where the branch should be placed. At that position a (branch-) point appears on the line. This point can be shifted along the line or be used for change of line routing as well.

Drawing branch line from terminalpoint to main line



Shifting the branch point



To *delete* a connection, just select the line at its starting- or target point and drag it away (disconnect it) from the function block somewhere to a free space in the program area. The related line will disappear.

Branches and kinks of lines can also be *fixed* in their position by pressing the right mouse button when the cursor is placed on such a point. A fixed point is marked by a little cross (×) on the line. To remove a fixed point repeat these steps. Objects can be moved in the area but the fixed point stays where it is.

ibaLogic checks automatically whether the data types of input and output match. If not, ibaLogic performs the action which has been defined under menu ↪*File* ↪*Program settings* ↪*Conversions*. ibaLogic provides "Autorouting", i.e. if a function block is shifted, all of his connections will be shifted together with it. If needed, the connection lines can be shifted manually (see above).

Function blocks with untyped input and output connectors (overloadable) will adopt the data types for the connectors as soon as they are connected with one source or target object with a declared data type.

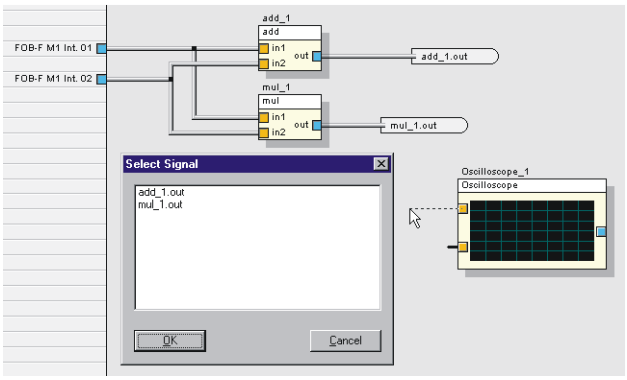
The other way round, these function blocks will loose their data type definition as soon as the last type-defining connection has been cut. At the same time all default values in these function blocks will get lost, because default values are only permitted when data types are defined.

3.9.2. IntraPage connectors (IPC)

An IntraPage connector (IPC) is a mean to simplify the diagrammatical representation – it's a replacement for a connection line. The use of IPCs is recommended if many objects on a page have to be connected or if long connection lines over several pages are required. The IPC can only connect objects which are located on the same hierarchical level, e.g. in one task or inside of a macro block. It's not possible to use IPCs for connections between objects on different levels, e.g. from the inside of a macro block to a function block outside of the macro in the program area.

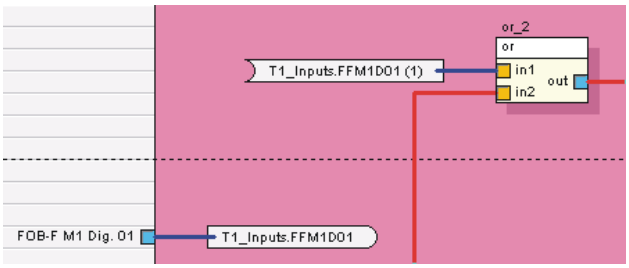
There are three ways to create an IPC:

1



Press the ALT key and draw a line from a starting point to a free space in the program area. The starting point for a signal source (for a "sending" IPC) is usually the output of a function block. To create the counterpart of a "sending" IPC (the "receiving" IPC), do it in the same way, starting at the target connection point (usually an input) and drawing the line "backwards" into a blank area. See example, left, at oscilloscope-block. If there are already IPCs in the program a selection list is displayed when creating a "receiving" IPC (e.g. *add_1.out* and *mul_1.out*). To connect, just select the desired IPC source and the connection is ready.

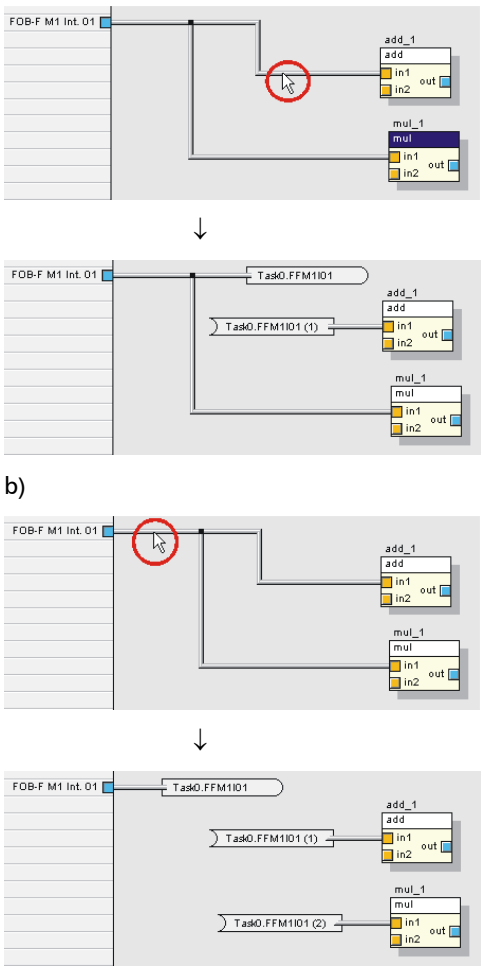
2



...or, by making a connection between two elements which are placed on different pages.

Connection lines which already exist won't be split up by dragging the function block over a page border.

3 a)



...or click on an existing connection line with pressed ALT key. The line, resp. the related network will be split up if acknowledged. Note, that branched connections will be replaced differently depending on the place where the IPC is defined. If the IPC is defined on a point-to-point connection or on a branch "behind" (in terms of data flow) the branching point without any further branches there will be just one sending and one receiving IPC (a).

If the IPC is defined "before" the branching point there will be one sending and as many receiving IPCs as branches (b).

The name of the IPC is given automatically depending on its origin. It could be *FUNCTIONBLOCK.CONNECTOR* or *TASKNAME.LABEL*. Of course,an IPC can be renamed by doubleclick on either the source part or the target part. Even the position and the size of the IPC can be changed. The size of the IPC can also be preset in the menu ↪*File* ↪*Program settings* ↪*Edit*.

The method to delete an IPC is the same as for connection lines by disconnecting the source, resp. the target point. If a signal source for a "sending" IPC is deleted the "sending" IPC itself and all corresponding target IPC will get lost as well. Target IPCs can be deleted individually.

A source-IPC as an object can only be deleted after all of its targets has been deleted.

3.9.3. Off-Task connectors and OPC-connections

OffTask Connectors (OTC) are used for inter-task communication whenever a connection between one or more tasks is required.

Creating an OffTask connector

1. Place the mouse cursor in blank space of the layout.
2. Open the menu ↪*Edit* ↪*New* ↪*Off-Task Connector* (or via context menu); the dialog as shown beneath will open.
3. If a new OffTask connector should be created please enter first a name into the field "Name". ibaLogic will give an error message and reject the name if a source connector of the same name is already defined.
   Some restrictions concerning the name may apply, please refer to chapter 7.2 for more information.
4. There are two methods to create a target connector:
   a) Select the source connector (click) and copy it to the clipboard then switch over to the task where the target connector should be placed and paste it. The OffTask connector will be pasted as a target connector automatically.
   b) Switch over to the task where the target connector should be placed and open the same dialog as described in step 2. In the dialog open the picklist in the field "Name", the desired source connector and uncheck the box "Output source".

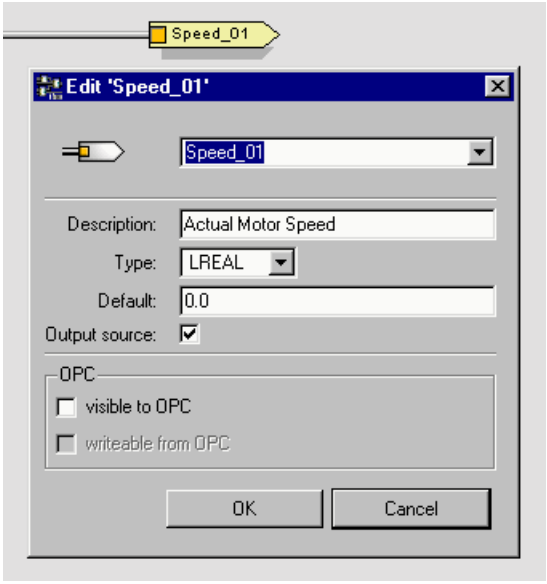
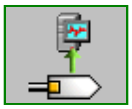


Fig. 45 OffTask connector, dialog

❑ ***Settings***

- *Description*: Entry of an explanatory comment. This description will also appear in the tooltip pop-up when the mouse cursor is placed on the connector of the OTC.
- *Type*: Selection of the desired datatype from a picklist.
- *Default*: Display or entry, respectively, of the default value of the OTC. After programstart the OTC will use this value. If the option "OPC-writing sets default values" in the menu ↪*File* ↪*Program settings* ↪*Edit* has been activated the default value of the OTC can be overwritten by an OPC-client, e.g. by a HMI system.
- *Output Source*: Check this box if the OTC is supposed to transmit data. When defining an input connector (target-OTC) uncheck this option.

Because OffTask connectors are the link to / from an OPC-interface there are two more options available:



- *OPC Visible*: ...when checked, this option enables the OTC to be visible in the browser of a connected OPC-client, OPC-icon in dialog changes (see left)
- *OPC->ibaLogic*: ...when checked this option allows an OTC (input / target-connector) to be written by an OPC-client.

Thanks to these options OTCs may be used for communication with HMI-systems. In that case ibaLogic is always OPC-server. OTCs, tagged as *OPC Visible* are visible in the browser of the OPC-client and can be selected for display.

If *OPC->ibaLogic* is activated a HMI system can send data to ibaLogic. The option *OPC->ibaLogic* can only be activated for target connectors which have no corresponding source-OTCs.

Inside one task an OTC can only exist one time, i.e. two or more "receiving" OTC with the same name in the same task are not allowed. (This is done with IPCs.)

"Receiving" OTCs, resp. target-OTCs can exist without a corrsponding source-OTC. The output of such an OTC is defined by its default value. Furthermore, target-OTCs without a source-OTC are represented by grey color in the diagram.

Because OTCs are objects they can be placed, deleted and altered in the usual way.

An OffTask connector has a dark grey color if it is neither declared as an output source nor connected to a data source. Else, it has a light grey color.

3.9.4. Switch and slider - smart helpers for testing

Switches are used for the online-operation of binary signals. On a right mouse-click the switch acts like a ON/OFF-switch (1st click = ON, 2nd click = OFF). On a left mouse-click the switch acts like a push-button (ON as long as mouse-button is down). The operation of analog values is performed by sliders which allow to alter a value continuously between MIN- and MAX-limits by shifting the slider knob with the left mouse button. For accurate adjustment (increments of 1/1000) click shortly in the slider field and then use the cursor keys ← / → on your keyboard. Both switch and slider will stay on their settings even after Stop / Start of the layout.

(see example below).

Example for use of switches and sliders

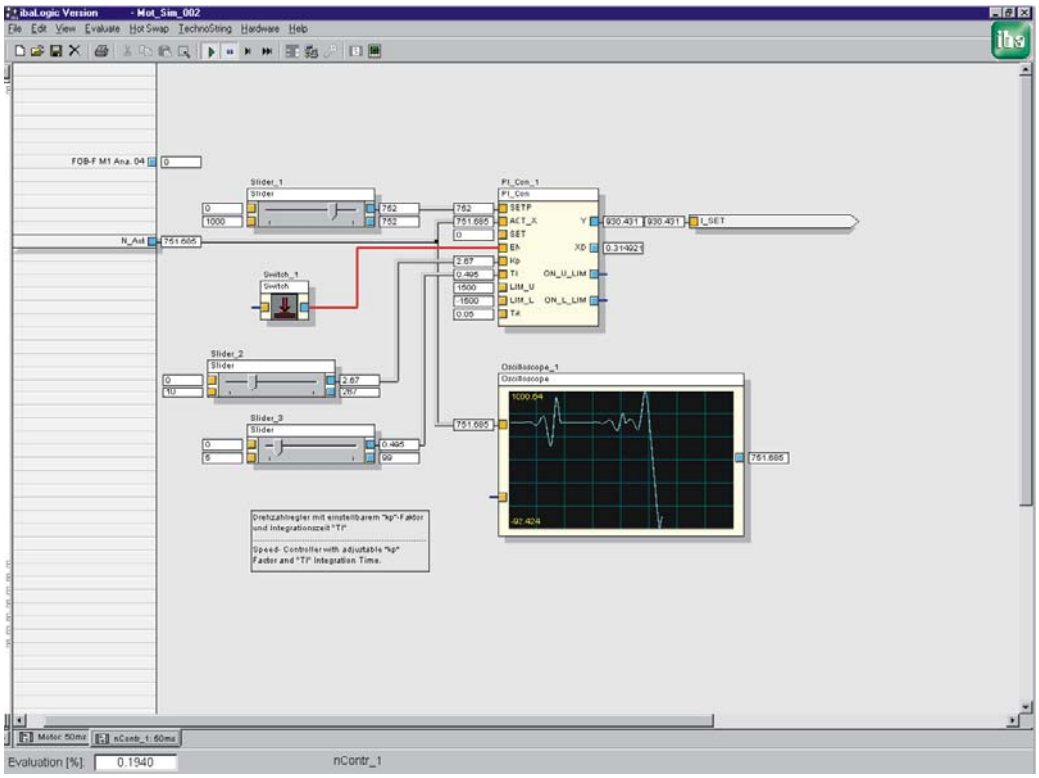
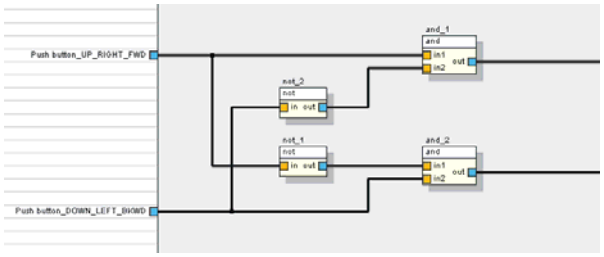
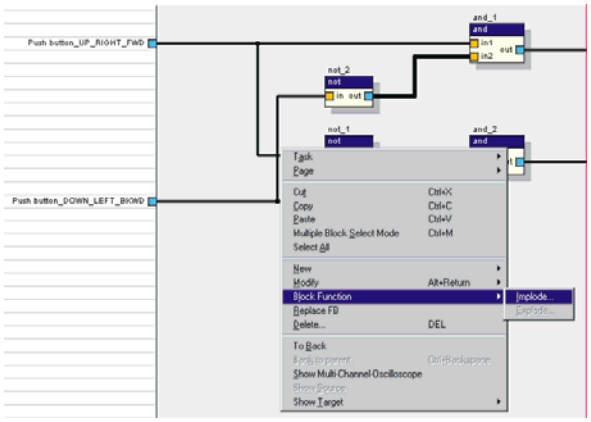
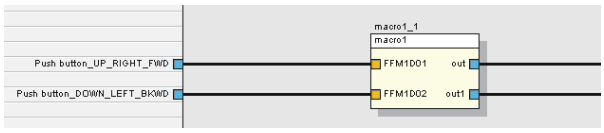
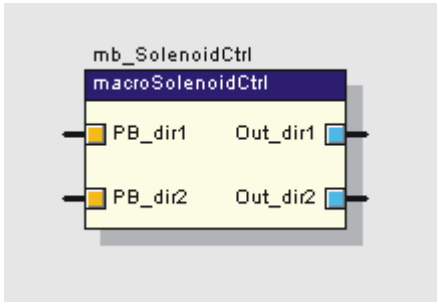


Fig. 46 Switch and Slider, sample application

3.10 Combining objects and creating macros

One of the outstanding features of ibaLogic is the easy way to cast a network of several objects, e.g. function blocks, and their connection lines into a new function block (macro block). This so called "Bottom-up design" feature is very useful for improving the clearity of a network or for reusing a complex function several times.

	As an example let's take a simple interlocking function as used for solenoid valve control. In order to avoid the use of these four function blocks tens of times in a project it is recommended to build a macro block. To combine the corresponding function blocks mark them by clicking the blocks with <SHIFT>-key pressed or using the multiple block selection mode (button in the tool bar).
	The function blocks and their connection lines are selected. Then press <SHIFT> and the right mouse button to get the edit menu and choose ↪ *Block Function* ↪ *Implode* and confirm the query.
	A new function block will be created. Doubleclick on the new block will open it for display and editing of the inner logical structure.
	In order to make this macro block independent from the former in- and outputs and to make it available for multiple use, the name, the in- and outputs should be renamed. This has to be done in the dialog which opens under menu ↪*Edit* ↪*Modify* ↪*Macro Block* (macro block must be selected)

On the other side, there is the possibility to create an empty macro block first and fill in the functions later (top-down design). For that, use the menu ↪ *Edit* ↪ *New* ↪ *Macro Block* and define the input- and output connectors of the macro block. Only these connectors will be available as inputs and outputs inside the macro block.

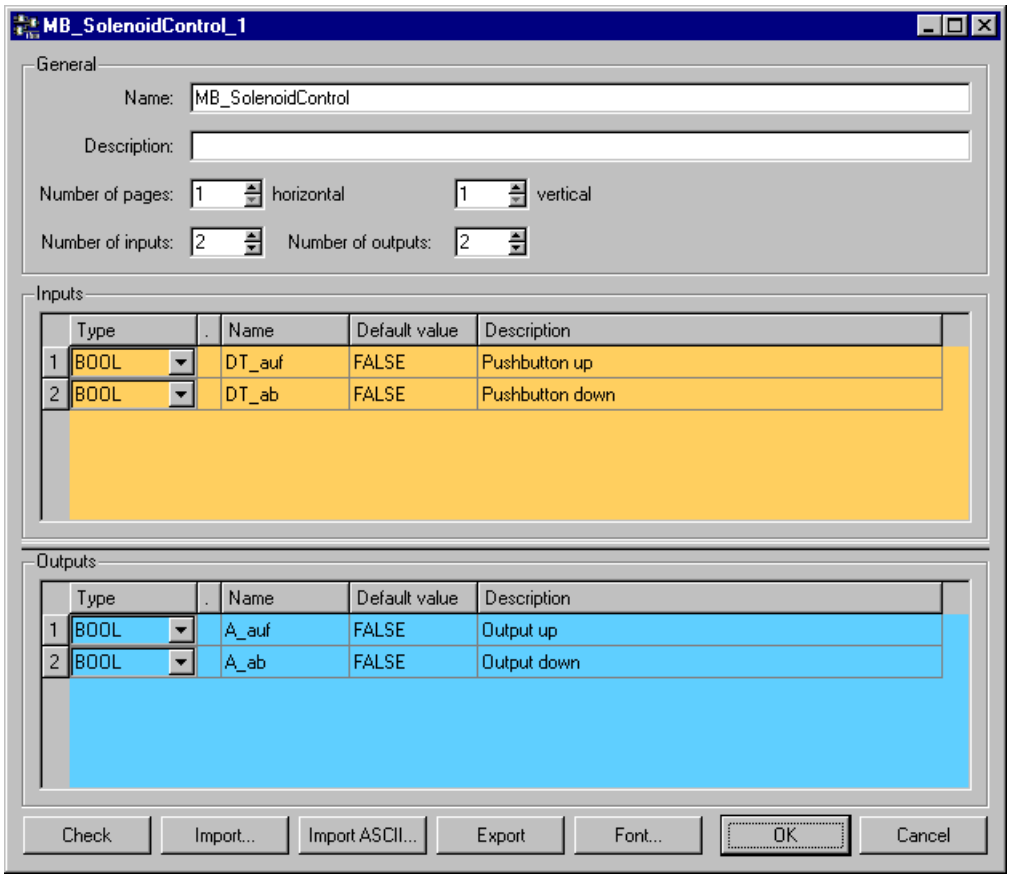


Fig. 47 Creating a macro block

To leave the macro level in the function block diagram click the right mouse button and choose ↪ *Back to parent* or press <Ctrl>+<Backspace>.

3.11 Creation of a new function block

ibaLogic possesses a large library of ready-to-use function blocks. (See 4). Though, the major part of problems can be solved with these function blocks it may be required to have a specialized function block for a particular solution. For that, ibaLogic offers two easy methods.

3.11.1. Creating a function block without Structured Text (ST)

Example: The new function block should return the difference of two input values on one hand and their average value on the other hand.

Open the function block window by means of the "Edit-menu" (↪*Edit* ↪*New* ↪*Function block*).

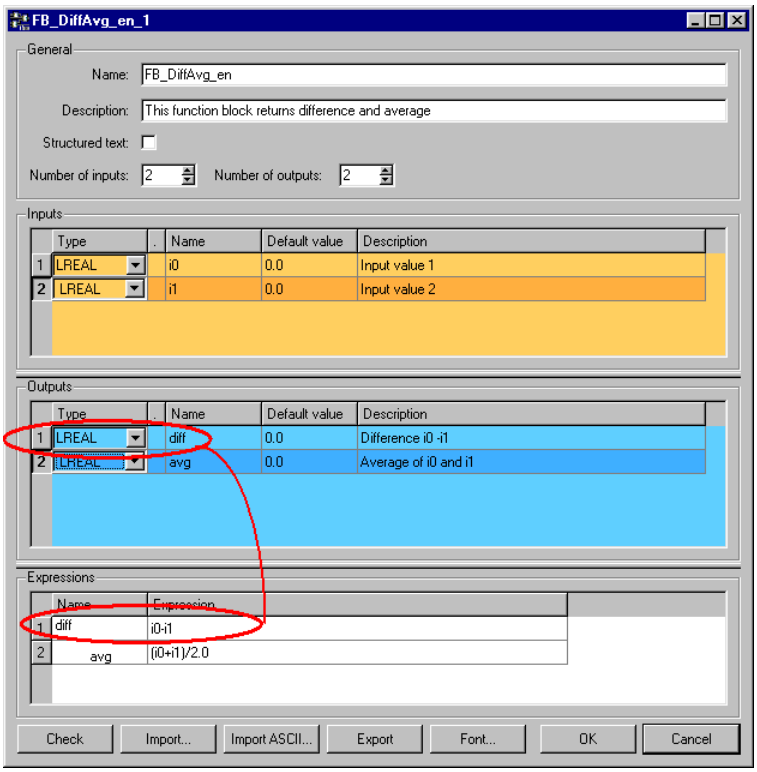


Fig. 48 Create a new function block

Start by modifying the entries in the following fields: *Inputs* to "*2*", *Outputs* to "2", *Name* to "*fb_example_1*", *Description* to "*This function block returns...*".

Notice that each time you add either inputs or outputs, new rows are added either on the corresponding yellow or blue space. If you decrease the number of inputs or outputs, then the rows are deleted after accepting the confirmation dialog. For each input click on the field in the column *Type* and select "*Real*" from the list of possible options, take a time to explore all types that you can use in the future. The default type "*LREAL*" comes from the system's general settings under menu ↪*File* ↪*Program settings* ↪*Edit Settings, Preset*.

The names of the input and output signals (i0, i1, o0, o1) may be renamed as well, if required, e.g. "diff" instead of "o0" and "avg" instead of "o1". To rename the signals just click in the corresponding fields in the table and overwrite the old name.

These are just examples for names to give. You may choose any name for your project layout. Be careful not to use reserved names as explained in the manual. If you do so, a warning message will appear and the entry will be rejected.

You can also change the default values, but this doesn't make sense for our example, keep it in mind for you future projects. Notice the buttons on the right side of the tables in the dialog that enable you to move or modify the selected row.

Now, you have to program the function block.

Assure that the Structured Text check box is unchecked and click in the row of the first output in the blue area. In the white "Expressions" area you'll find a table with all defined outputs. In this table, column "Expressions" you may enter all statements and expressions for the evaluation of the corresponding outputs. Use only simple mathematical expressions, formulas or assignments, as shown in the example or in the table below. For the second output do the same accordingly.

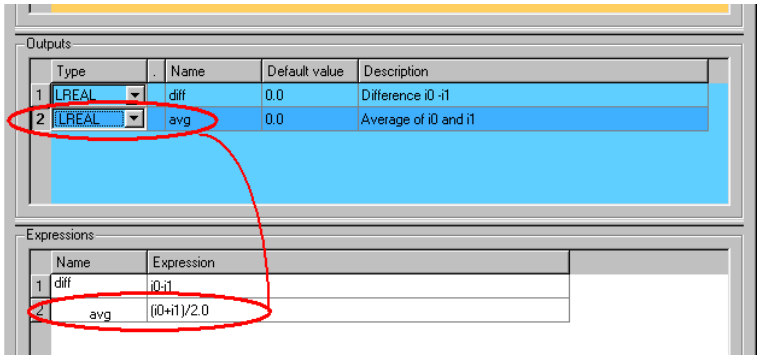


Fig. 49 Create FB without ST

3.11.1.1. Operations for simple FB-creation

Operation	Example	Result of example	Description	Priority
()	(2+3) * (4+5)	45	Brackets	highest
**	3**4	81	Power	
-	-10	-10	Negation	
NOT	NOT DIG01	FALSE (if DIG01=TRUE) TRUE (if DIG01 = FALSE)	Inversion	
*	10*3	30	Multiplication	
/	6/2	3	Division	
+	2+3	5	Addition	
-	4-2	2	Subtraction	
<, >, <=, >=	4 > 12	FALSE	Comparison	
&, AND	TRUE & FALSE	FALSE	Boolean AND	
XOR	TRUE XOR FALSE	TRUE	Boolean Exclusiv OR	
OR	TRUE OR FALSE	TRUE	Boolean OR	lowest

Table 5 Operations for simple FB-creation (no ST)

You may check the correct programming of your operation by pressing the "*Check*" button.

When done, press the "*OK*" button and place the function block in the diagram.

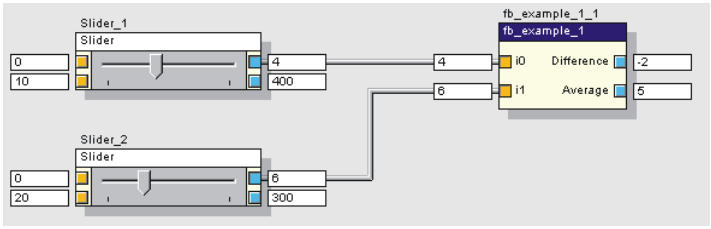


Fig. 50 Placement of new FB

If you click the "Export" *button before clicking the* "OK" *button, then the FB you just created will be available for drag and drop use on the* "Local FBs and Macros" *folder in the* "Functions" *tree. This is really useful for large projects. The FB is physically stored as .fbm file under* \ibaLogic\configuration\FBs_Macros *folder on your hard drive. You can share with more people your FBs by copying this file or by sending it by e-mail. When somebody share an FB with you, copy the .fbm file in the \FBs_Macros folder on your hard drive before starting ibaLogic*

3.11.2. Creating a function block with Structured Text (ST)

The same principle as decribed before applies. But in order to program a function block in ST you must check the "Structured Text" check box in the function block dialog window.

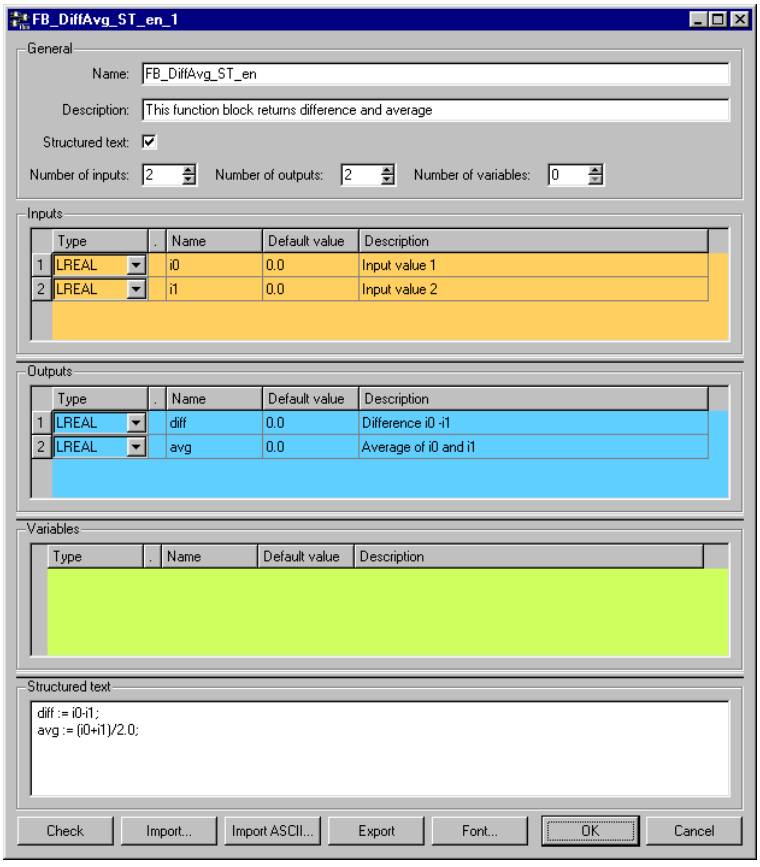


Fig. 51 Create FB with ST

Now, there is only one program code for the entire function block and no individual output assignment as before.

First, let's have a look on some basic terms and elements of ST.

3.11.2.1. Operations and statements in Structured Text (ST)

Programs written in ST look very much like those programs written in PASCAL. In ST a statement is terminated by a smicolon. Comments are marked with *(** at the beginning and ***)* at the end. Data are processed by expressions and statements. Expressions consist of operations (see table below) and operands and they deliver a result. Operands can be literals, variables, other expressions and function calls.

Operation	Example	Result of example	Description	Priority
()	(2+3) * (4+5)	45	Brackets	highest
**	3**4	81	Power	
-	-10	-10	Negation	
NOT	NOT DIG01	FALSE (if DIG01 = TRUE) TRUE (if DIG01 = FALSE)	Inversion	
*	10*3	30	Multiplication	
/	6/2	3	Division	
MOD	MOD (17,10)	7	Modulo (Divisionsrest)	
+	2+3	5	Addition	
-	4-2	2	Subtraction	
<, >, <=, >=	4 > 12	FALSE	Comparison	
=	T#26h = T#1d2h	TRUE	Equal	
<>	8 <> 16	TRUE	Not equal	
&, AND	TRUE & FALSE	FALSE	Boolean AND	
XOR	TRUE XOR FALSE	TRUE	Boolean Exclusiv OR	
OR	TRUE OR FALSE	TRUE	Boolean OR	lowest

Table 6 Operations in ST

3.11.2.2. Data declarations in Structured Text (ST)

In ST-statements the datatypes UDINT and DWORD shall be marked with "#" (e.g. UDINT#0, DWORD#0) in order to distinguish between them and signed INTEGER-variables. Constants on base 16 (hex) are declared by "16#" (e.g. 16#2BC1F9) and they are automatically considered as DWORD. Constants on base 2 are declared by "2#" and those on base 8 by "8#". Time variables are declared by "T#" supplemented by "d" (day), "h" (hour), "m" (minute), "s" (second) and "ms" milli second (e.g. T#67d12h17m42s).

Data declarations in ST (represented as text):

```
VAR_INPUT
  in_bool:    BOOL   := FALSE;
  in_int:     INT    := 0;
  in_dint:    DINT   := 0;
  in_udint:   UDINT  := UDINT#0;
  in_dword:   DWORD  := DWORD#0;
  in_real:    REAL   := 0.0;
  in_lreal:   LREAL  := 0.0;
  in_time:    TIME   := T#0ms;
  in_string:  STRING := '';
END_VAR
```

```
VAR_OUTPUT
  out_bool:   BOOL   := FALSE;
  out_int:    INT    := 0;
  out_dint:   DINT   := 0;
  out_udint:  UDINT  := UDINT#0;
  out_dword:  DWORD  := DWORD#0;
  out_real:   REAL   := 0.0;
  out_lreal:  LREAL  := 0.0;
  out_time:   TIME   := T#0ms;
  out_string: STRING := '';
END_VAR
```

3.11.2.3. Statements in Structured Text (ST)

Statement	Example	Descriptions
RETURN	RETURN;	Go back, immediately abort function block
IF	IF a < b THEN c:=1; ELSIF a = b THEN c:=2; ELSE c:=3; END_IF;	Comparison, selection
CASE	CASE f OF 1: a:=3; 2: a:=4; ELSE a:=0; END_CASE;	Selection
FOR	FOR a:= 1 TO 10 BY 2 DO f[a] := b; END_FOR;	loop (unconditional)
WHILE Not supported	WHILE b > 1 DO b := b/2; END_WHILE;	loop (conditional) Not supported, risk of endless loops
REPEAT Not supported	REPEAT a:= a * b; UNTIL a > 10000 END_REPEAT;	Repetition Not supported, risk of endless loops
SET_VALID	SET_VALID (<Variable name>, FALSE)	Set a variable valid / invalid (z.B. FB-Anschluss)
SET_DEFAULT	SET_DEFAULT (<variable name>, <value>)	Set a default value of a variable
ARRAY-access	<variable name>[i] <variable name>[i,j,k,m]	Access on an one-dimesional array Access on an four-dimesional array
EXIT	EXIT;	Immediately abort function, e.g. in FOR-loops

Table 7 Statements in ST

Please note that FBs can not be used in ST-statements or operations. Only functions are allowedto be used.

Moreover, some restrictions apply concerning the usage of names for FBs, or functions which are reserved by ibaLogic, see chapter 7.2.

3.11.2.4. Function block PT1 in Structured Text (ST)

In the following a delay element of 1st level (PT1) is taken as a model for the creation of a function block.

The mathematical definition of a PT1 element is:

$$Y = Y_{n-1} * e^{-(TA/T1)} + X1_{n-1} (1 - e^{-(TA/T1)})$$

with:

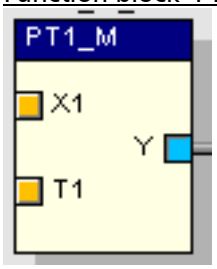
Y	=	Output value of PT1-element
Y_{n-1}	=	Output value of the previous program cycle
X1	=	Input value
$X1_{n-1}$	=	Input value of the previous program cycle
T1	=	Delay time [sec], output value is about 63% of input value
TA	=	Scan time [sec]

A variety of function blocks for regulation require the task scan time and the lapsed time since start of the application. These time values are made available by the global variables in ST:

g_EvalDeltaTime = time lapsed since last start of the task; the use of this variable will help to eliminate deviations in scan time and to evaluate the correct results.

g_EvalTime = time lapsed, since start of the application

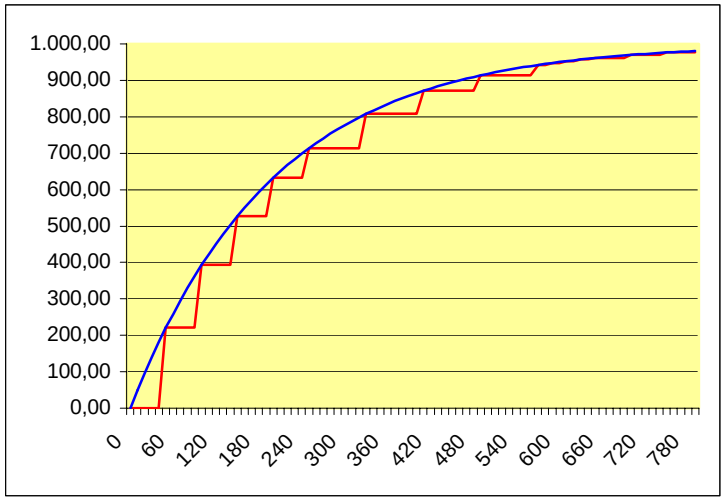
Function block "PT1_M"



Programmcode "PT1_M" in Structured Text

```
TA_T1:=-time_to_lreal(g_EvalDeltaTime)/T1; (* Get Cycletime
TA and evaluate –TA/T1 *)
E_TA_T1:=2.71828**TA_T1; ; (* Evaluate e** TA/T1*)
Y:=Y * E_TA_T1+(X_N1*(1.0-E_TA_T1)); (* Y- calculation *)
X_N1:=X1; (* Copy X1(n-1) = X1 *)
```

Regulator output PT1-function block
blue: scan time const. 10ms
red: scan time const. 50 ms



To create a function block use the menu ↪ *Edit* ↪ *New* ↪ *Function Block...*

In the new dialog window there are five areas as follows:

❑ ***General***

Definition of number of in- and outputs, function block name and description

❑ ***Inputs***

Definition of input variables with data type and description

❑ ***Outputs***

Definition of output variables with data type and description

❑ ***Variable***

Definition of block-internal variables with data type and description

❑ ***Definition***

Structured Text (ST) statements and expressions

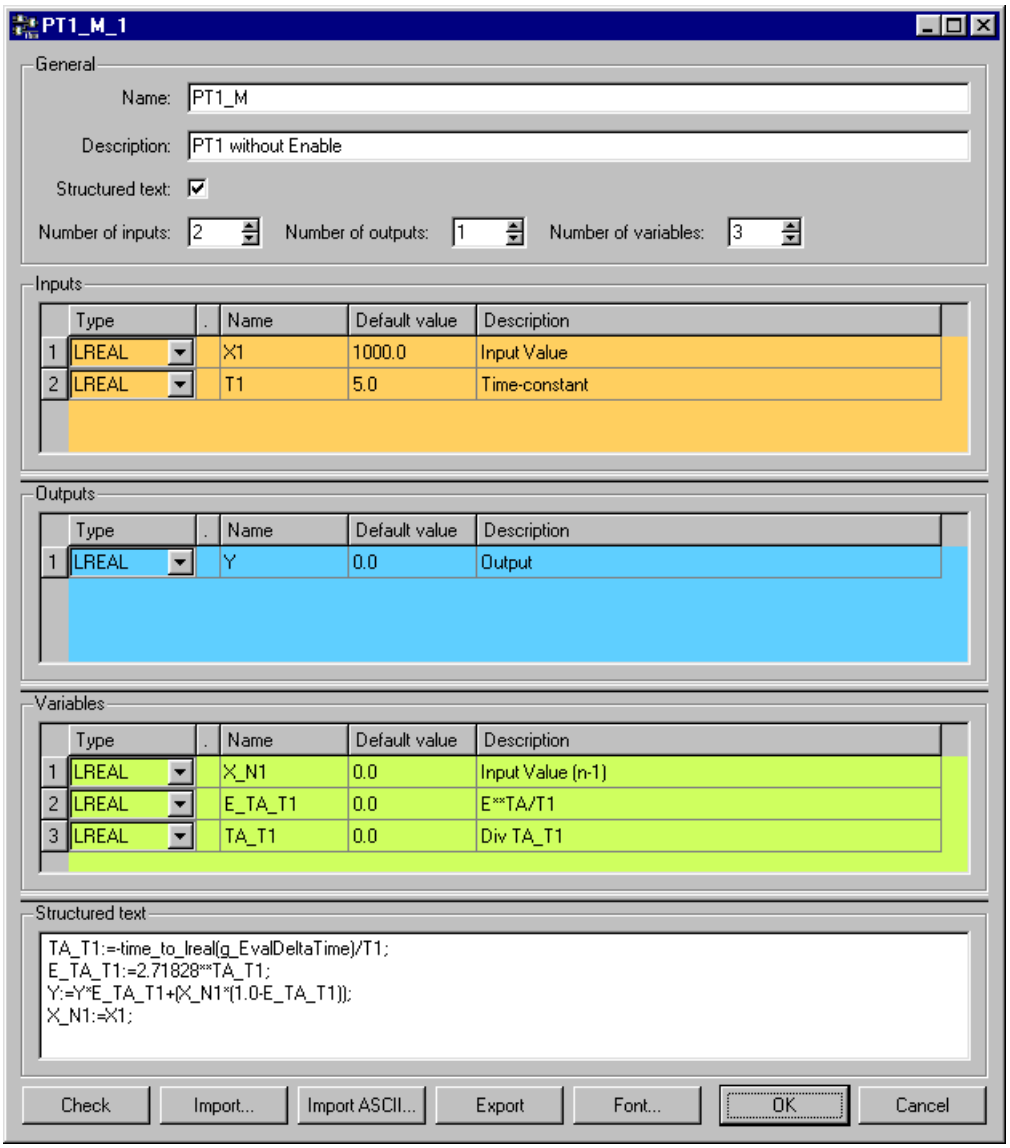


Fig. 52 Create function block PT1M

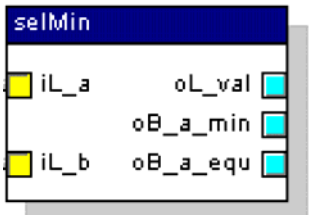
3.11.3. Examples for statements in Structured Text (ST)

The following examples show the essential statements in ST (if, case, for etc.), used for function blocks.

3.11.3.1. IF- and ELSIF-statement

The function block to be created "selMin" should always return at the ouput "val" the lower value of either of its two input values "a" or "b". If the input "a" is lower or equal to "b" then the boolean output "a_min" is set TRUE. If inputs "a" and "b" have the same value then the LReal-output "val" is set to 0.0 and the boolean output "A_equ" is set TRUE.

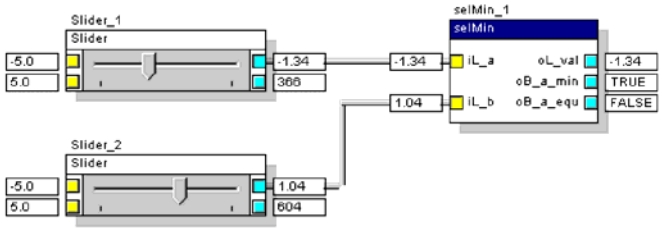
Function block "selMin"



Program code "selMin" in Structured Text

```
oB_a_equ:=FALSE;      (* Default setting *)
if iL_a=iL_b
  then oL_val:=0.0;  (* a=b, Output=0.0 und equ=TRUE *)
  oB_a_min:=TRUE;
  oB_a_equ:=TRUE;
  elsif iL_a<iL_b       (* Check a<b *)
    then oL_val:= iL_a; (* a is smaller, Output to val *)
    oB_a_min:=TRUE;     (* a_min = TRUE *)
    else oL_val:= iL_b; (* b is smaller, Output to val *)
    oB_a_min:=FALSE;    (* a_min = FALSE *)
  end_if;
```

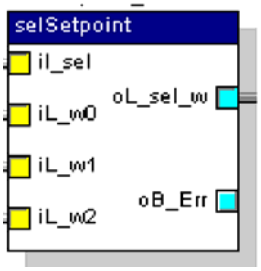
Application model "selMin"



3.11.3.2. CASE-statement

The function block to be created "selSetpoint" should return at the output "sel_w" one of the three LReal-inputs "w0", "w1" or "w2", selected by the value of INT-input "sel". If the input "sel" is not equal {0, 1, 2}, the ouput "sel_w" will be set to 0.0 and the boolean output "Err" will be set TRUE.

Function block "selSetpoint"



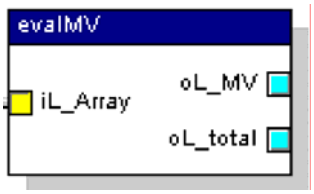
Program code "selSetpoint" in Structured Text

```
oB_Err := FALSE;          (* Default setting *)
CASE iI_sel OF            (* CASE- selection 0,1,2 *)
0: oL_sel_w:=iL_w0;       (* CASE = 0 *)
1: oL_sel_w:=iL_w1;       (* CASE = 1 *)
2: oL_sel_w:=iL_w2;       (* CASE = 2 *)
ELSE oL_sel_w:=0.0;       (* value iI_sel unequal 0,1,2 *)
oB_Err:= TRUE;            (* sel_w = 0.0, Err = TRUE *)
END_CASE;
```

3.11.3.3. FOR-statement

The function block to be created "evalMV" should return the total sum and the average of an array of 16 LReal variables. The function block uses the internal variable "count" for counting.

Function block "evalMV"



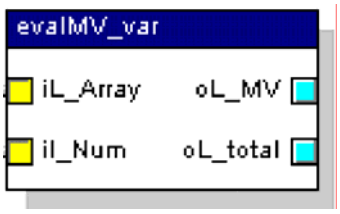
Program code "evalMV" in Structured Text

```
oL_total:=0.0;                          (* Default setting *)
FOR count:=0 TO 15 DO                     (* FOR- 0 to 15 *)
  oL_total:= oL_total+iL_Array[count]; (* total-value *)
END_FOR;
oL_MV:=oL_total/16.0;          (* Mean Value evaluation *)
```

3.11.3.4. EXIT- and RETURN-statement

The previous created function block "evalMV" has been renamed in "evalMV_var" and supplemented with a further INT-input "iI_Num". This additional input defines the range for sum and average evaluation in the array, e.g. 7 = sum and average of array-elements no. 0 ...7. By the mean of the "EXIT" statement in the IF-query, the FOR-loop will be terminated before reaching its limits, but the average value will still be evaluated. Using the "RETURN" statement instead of "EXIT" will terminate the function immediately without evaluation of the average value.

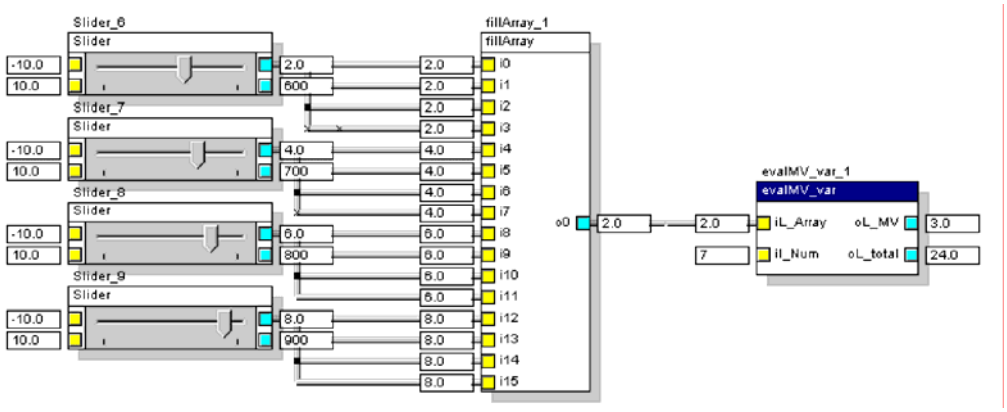
Function block "evalMV_var"



Program code "evalMV_1" in Structured Text

```
oL_total:=0.0;                          (* Default setting *)
FOR count:=0 TO 15 DO                     (* FOR- 0 to 15 *)
  oL_total:= oL_total + iL_Array[count]; (* total-value*)
    IF count>(iI_Num-1)   (* max. Number Input reached *)
    THEN EXIT;     (* or RETURN; FOR- Loop termination *)
    END_IF;
END_FOR;
oL_MV:=oL_total/int_to_real(iI_Num+1);  (* Mean Value *)
```

Application model "evalMV_var"



3.12 Creating your own DLL

Creating macros and function blocks with ST are very simple ways to solve many problems of automation. But as easy it is to create them as easy is it to copy them and to unserstand their manner of working.

Sometimes you may prefer a less open prove of your engineering expertise, e.g. in case of a sophisticated technological solution but you want to prevent the cheap distribution of your know-how.

In such a case the possibility of creating DLLs which include your brain's work in a compiled form so that no one can figure out your tricks is a real advantage.

3.12.1. C-Compiler

For writing and compiling the DLLs we've tested and approved the following C-compilers:

- ❑ Microsoft Visual C++ 5.0
- ❑ Microsoft Visual C++ 6.0
- ❑ Other, such as Borland are supported too

3.12.2. Source files needed for creating DLLs

The following source files which come with the ibaLogic installation CD-ROM are required:

- *name*dll.cpp: contains the Procedures and the DLL – Body;
  the user may add inputs, outputs or make changes in the Procedures InitEvaluation, Evaluate, ExitEvaluation
- *name*dll.def: contains the Assignment between DLL Procedures and Numbers; the library name must match the DLL Name !!!
- dllForm.hpp: contains the interface definition;
  no changes neccessary.

↗ *Refer also to chapter **7.1**. There you'll find the program lisitngs of the "sampleDLL" which is delivered along with ibaLogic.*

3.12.3. Procedure for creating new DLLs

For creating new DLLs it is recommended to use the simpleDLL frame:

1. Create a new DLL project with your own DLL name.
2. Copy the simpleDLL.cpp, simpleDLL.def and DLLform.hpp files into your project directory.
3. Rename the simpleDLL.cpp and simpleDLL.def files according to your own DLL name.
4. Change the library name in the ....DLL.def file according to your own DLL name.
5. Add the .cpp, .def and .hpp file to your project.
6. Build the new DLL.
7. Copy the new DLL into your IBALogic directory.

3.12.4. Frequent obstacles

- ❑ Don’t forget to add the .def File to your project, else the DLL won’t work with IbaLogic.
- ❑ You may add and use variables to your DLL or Evaluate Procedure, but if you use more than one instance of the same DLL you must save the data used between two calling cycles in the dynamic data area. Otherwise the variable exits only once for all instances.
- ❑ In case you want to calculate some cycle time-dependent functions you should use the variable "pGlobal" which is a pointer to a relative time variable.
- ❑ The DLL runtime will be added to the cycle time of the task which is calling the DLL.
- ❑ It is recommended to use threads for time consuming functions.
- ❑ Function blocks should use the "invalid flag" and they should write data to the periphery only if the "online flag" is set.
- ❑ For the purpose of testing a DLL ibaLogic may be started as the executing program.
- ❑ The interfacing functions of a DLL will be called directly from ibaLogic.
- ❑ Not all programming errors which may be included in a DLL can be detected and cushioned by ibaLogic. Hence, these errors may even cause a crash of ibaLogic.

3.12.5. Linking the DLL in ibaLogic

ibaLogic uses the following calls in conjunction with the function block interface:

Call	Function
GetInstanceDynamicDataSize()	Query for fixing the size of dynamic data
GetDllDescription()	Query of description of the DLL
GetCount()	Query of number of inputs and outputs
GetName()	Query of name of each input and output
GetDescription()	Query of description of each input and output
GetType()	Query of datatype of each input and output
GetArrayHeader()	Query of array datatype of an input or output
GetDefaultValue()	Query of default value of an input or output

The following call will be needed in runtime:

Call	Function
InitEvaluation()	Single call at start of evaluation; used for initialization
SetInputValue()	Cyclic call for every input once per cycle prior to each evaluation.
Evaluate()	Cyclic call once per cycle
GetOutputValue()	Cyclic call for every input once per cycle after each evaluation.
Exit Evaluation()	Single call at end of evaluation; used for cleanup

3.13 Testing and debugging of projects

ibaLogic offers several tools for the purpose of testing function blocks and more complex networks.

3.13.1. Single and multiple step mode, halt the project

If a project is being evaluated it could be switched into single or multiple step mode. This is useful in order to test sheer logical functions (sequences). There are the following corresponding buttons in the tool bar:

 (from left to right: Start/Stop Evaluation, Pause Evaluation, Evaluate 1 step, Evaluate n steps)

In order to switch in single or multiple step mode first press the pause-button.

If the project is running online (purple background color) there won't be no refresh of the external resources (in-/outputs) for the time between two steps! This means that the values will stay as they are what may have an unpleasant impact on the process.

In this case there is a risk of hazard for life or machinery!

The number of steps to be evaluated at one click of "multiple step" can be adjusted from 2 up to 64 steps in the menu ↪*Evaluate* ↪*Set Multiple Step Count*.

3.13.2. What to do, if values become sporadically invalid?

During evaluations it may occur that output values of function blocks become invalid due to bad starting conditions, division by 0 or limit violations.

Such invalid states are indicated in ibaLogic by a red cross in the output "terminal" of the function block, a red frame for the output value display and – if running in evaluation mode – the red representation of the value itself.

What could cause a variable to become invalid?

- ❑ Invalid real value
- ❑ Division by zero
- ❑ Intended setting of the "Invalid-bit" with *set_valid(<variable name>, FALSE)*
- ❑ Assignment of array elements if the index limits are violated
- ❑ Assignment of expressions, which contain already invalid values
- ❑ Being part of a chain or loop and depending on other variables in the same chain which are invalid.
- ❑ Input resources if the corresponding PC-board (FOB IO, FOB 4i) is not available or not alive and if the option "*Unavailable signals are invalid*" has been selected in the system settings.

1. *Note: Arrays have exactly one valid-flag. If one element in the array is invalid, so is the entire array.*
2. *If a variable becomes invalid, the last valid value is preserved.*
3. *If an invalid variable occurs in a diagrammatical feed-back branch, there are several measures available for trouble-shooting or correction:*
   - *Start/Stop Evaluation or*
   - *Breaking up the logical network (delete a connection line, insert a function block etc.) or*
   - *insertion of a "set_valid"-statement [set_valid(<variablenname>, TRUE)] into the logical network; by that it's possible to configure a project in that way that it can fix the problem automatically in case of occurrence.*

Of course, the first two posibilities require that the variable won't stay invalid forever.

3.13.3. The ordinary oscilloscope for testing

This oscilloscope is designed for a swift check of a signal shape. It is to be placed like a function block and it displays immediately the connected signal. The input is of datatype "untyped". Arrays can not be connected to the oscilloscope (see also next chapter). There are no scales in order to survey the signal.

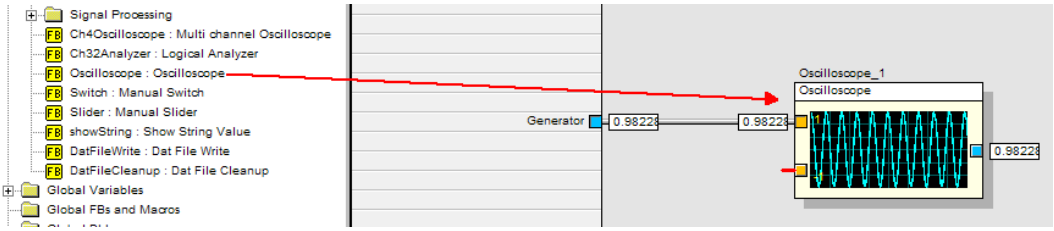


Fig. 53 Simple oscilloscope

3.13.4. The Multichannel Oscilloscope and Logical Analyzer

These two function blocks are very similar to each other and base on the same principle but they are used for different purposes.

- ❑ The logical analyzer (Ch32Analyzer) is a tool for display of up to 32 boolean signals simultaneously (only datatype BOOL is allowed).
- ❑ The multichannel oscilloscope is used for display of up to four signals in order to survey the signals, to optimize closed loop controls or to represent arrays (vectors, e.g. a FFT-result).

3.13.4.1. Usage

One of these function blocks should be placed in the function block diagram. Connected to the signals, it can be considered like a probe without display. Unlike the ordinary oscilloscope these function blocks use no extra processor time for graphic display as long as it is not enabled. So, it is possible to place and connect several oscilloscope-function blocks in the diagram without requiring excessive computing time for display except for the one which is activated. Only one instance of the oscilloscope (or logical analyzer) can be displayed at a time. Switching over to another display is easily done by pressing the tabs at the bottom of the display window.

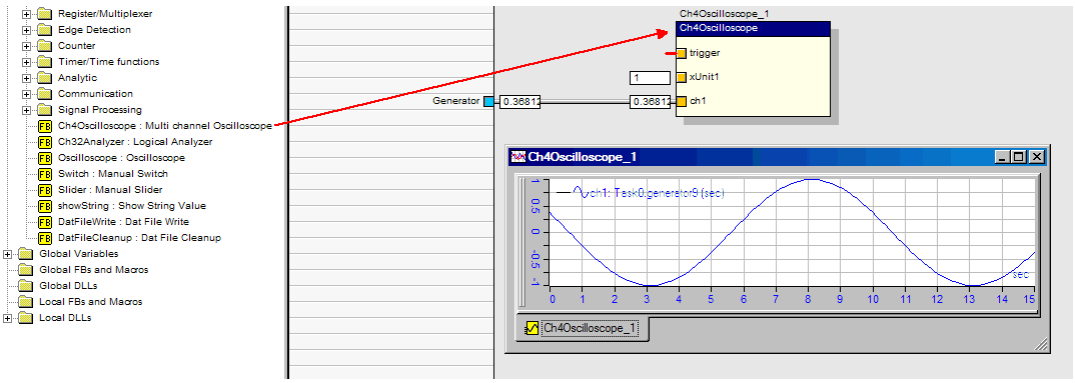


Fig. 54 Multichannel oscilloscope

The number of (input) channels can be adjusted after doubleclick on the function block.

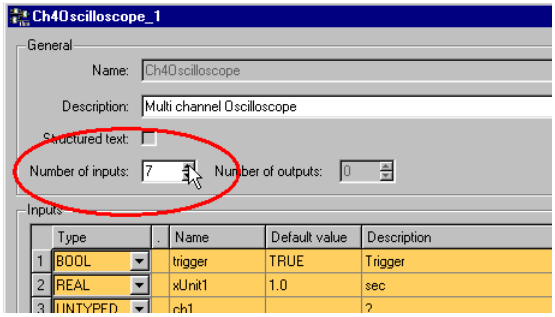


Fig. 55 Adjusting channels for multichannel oscilloscope

In case of the logic analyzer each additional channel means one binary input (BOOL) more. In case of the multi-channel-oscilloscope every additional input means one signal input (*ch*), always together with a scaling input (*x Unit*). By means of the latter input type, the display of each channel can be scaled individually. If a scaling input is not connected the corresponding channel will be scaled like the previous channel.

As long as the "trigger"-input is TRUE, the display is continuously refreshed. "trigger"-input = FALSE freezes the display.

Scaling the inputs

The default value for each scaling input is 1. Every value (e.g. real value or array element) is scaled by this quantity. For array elements this might be more complex if another base than the basic cycle time is needed.

↗ *See also box Dynamic scaling on page 3-42*

3.13.4.2. Operation

In order to open the display select the function block (Ch4Oscilloscope or Ch32Analyzer), make a right mouseclick and choose ↪*Show Multi-Channel-Oscilloscope* in the context menu or just click on the toolbar button  .

Use the same commands to show the logical analyzer display.

The user interface and the operation of the oscilloscope has been improved since ibaLogic version 3.86. The new operational concept resembles the one of ibaAnalyzer which is already well known by many users.

Different coloring of the curves, continuous compressing and stretching of the X- and Y-scales as well as the shifting of signals, respectively the combination of several signals in one signal strip and finally the measuring of values by means of rulers are available features.

Context menus are available in the areas of X-axis, Y-axis, graph and signal name for the corresponding settings.

Zooming is possible by holding the left mousekey depressed when drawing a frame in the graph. The command *Autoscale* in the context menu of a graph zooms out completely. To open the context menu just make a right mouseclick in the graph of the oscilloscope. Some options and settings are offered in the context menu concerning the display of signals.

A zooming in steps is possible with the context menus of X- and Y-axis.

The X- and Y-axis may be shifted when pointing on the scales, holding the mousekey depressed and move the mouse.

Using the *Autoscale* function may be helpful when a signal is not visible because it's out of scale.

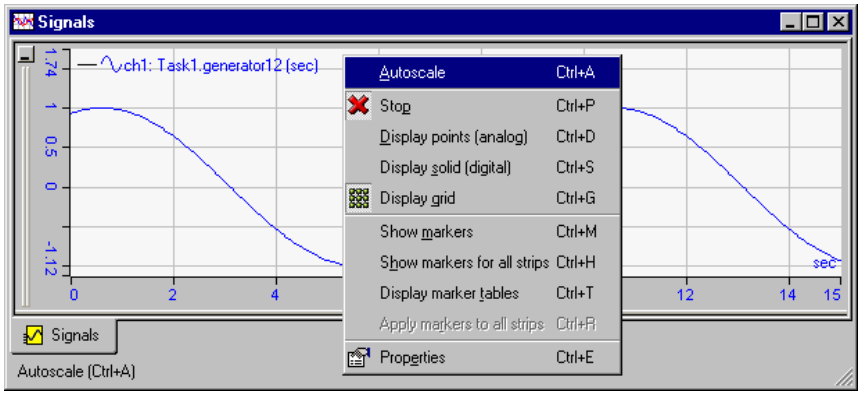


Fig. 56 Multichannel oscilloscope, autoscale

A simple measurement of the graphs is provided by two markers which can be activated via the context menu. The refreshing of the graph must be halted for that.

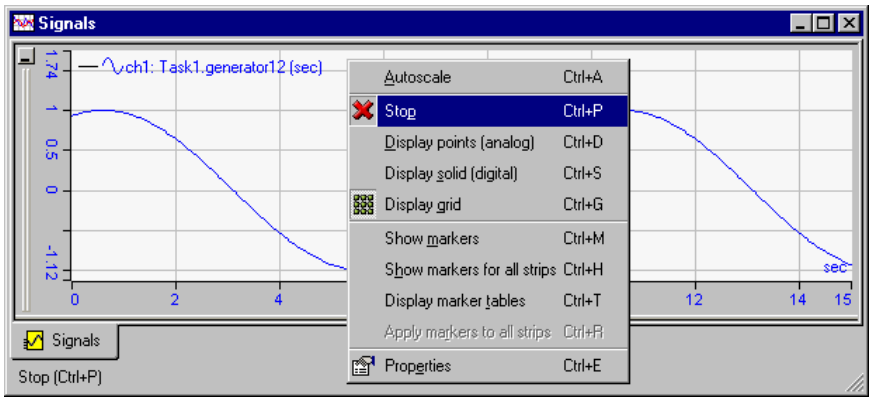


Fig. 57 Multichannel oscilloscope, stop refreshing

In order to see the values you should select ↪*Display marker tables* in the context menu. A table with the Y-values of all displayed graphs related to the X-position of the markers and their differences will open below the graphs.

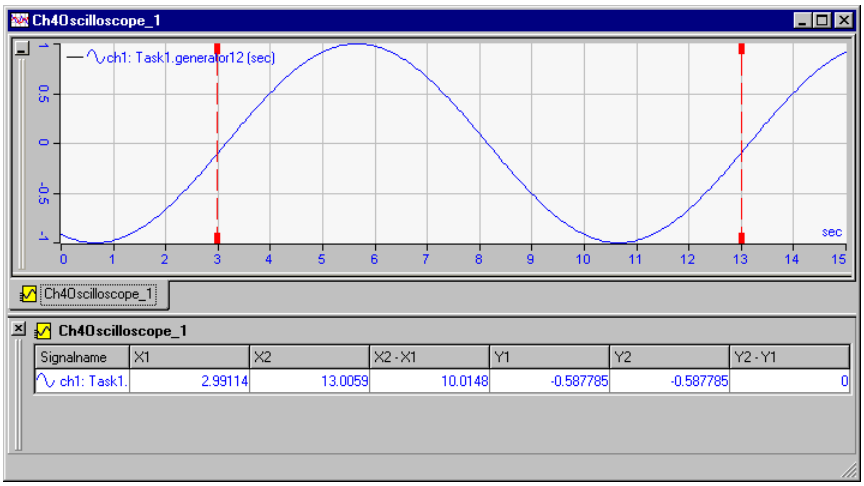


Fig. 58 Multichannel oscilloscope, rulers and data table

Each channel has its own graph if more than one channel of the multichannel oscilloscope is used. Each graph has its own X-axis and –scale (corresdonding to input xUnit).

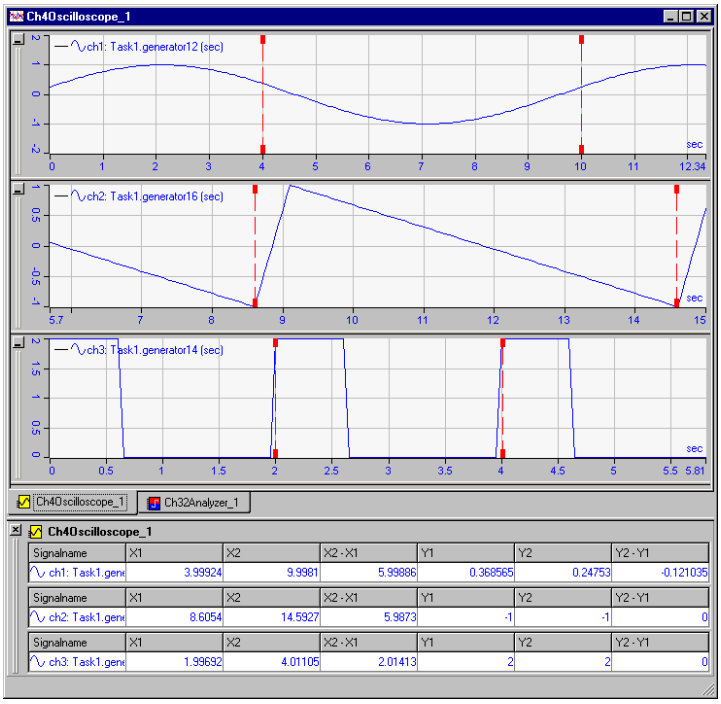


Fig. 59 Multichannel oscilloscope, multiple channels

The markers will be visible in all graphs after choosing ↪*Show markers for all strips* in the context menu. They may be moved indepently. If you like to have them all in the same position then choose a pair of markers in one strip as a reference pair. After the markers have been positioned open the context menu in the same strip and select ↪*Apply markers to all strips*.

Finally, several signals may be displayed together in one signal strip, just like in ibaAnalyzer. Place the mouse cursor on a signal name until it changes its shape (little waveline).

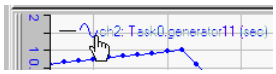


Fig. 60 Multichannel oscilloscope, move signal

Then drag the signal (mousekey depressed) to the target strip where the signal should be displayed.

Drop the signal somewhere in the strip: the signal gets its own Y-axis.

Drop the signal close to another signal name, as soon as a little arrow appears: the signal is assigned to the same Y-axis as the existing signal.

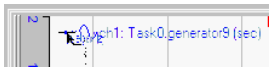


Fig. 61 Multichannel oscilloscope, gather signals

For a better distinction of the different curves paint them in different colors by ↪*Auto-color* in the context menu on the signal names in the strip.

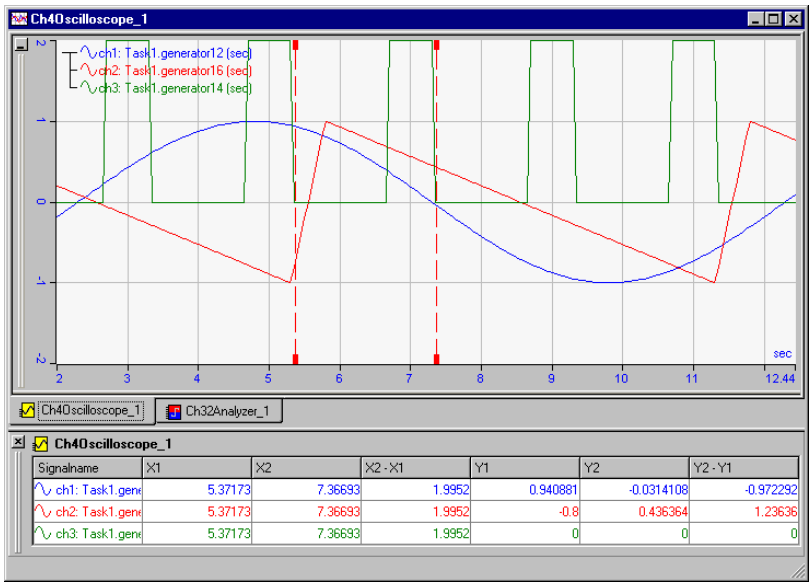


Fig. 62 Multichannel oscilloscope, automatic color

Now you can measure the signals by means of the markers.

The logical analyzer works in the same way, but there is no Y-axis because the values of the digtal signals can only vary between TRUE (1) and FALSE (0).

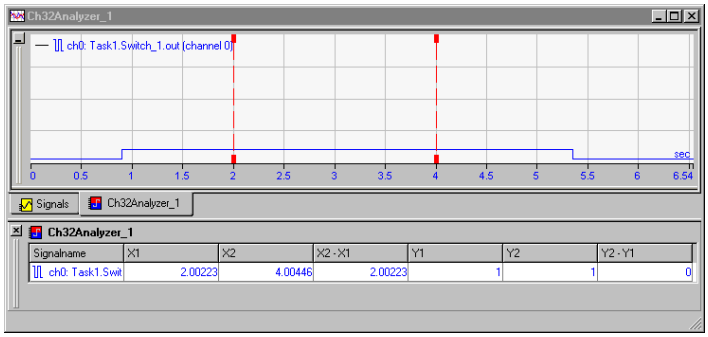


Fig. 63 Multichannel oscilloscope, CH32Analyzer

3.13.4.3. Sample application for multichannel oscilloscope and rfft function block

The following example shows how to use a multichannel oscilloscope in conjunction with a rfft function block. Please note that there is an input channel of ARRAY-type and that the scaling (xUnit) of both the time axis and the frequency axis (FFT) as well is evaluated dynamically.

Function block diagram

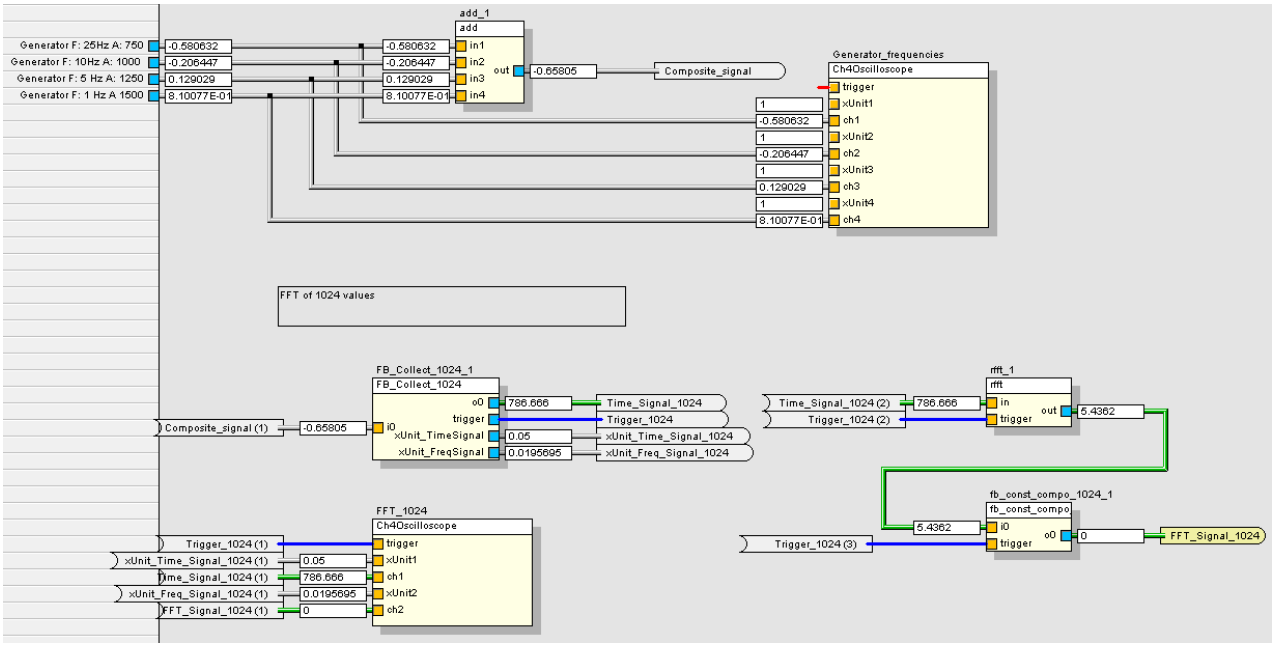


Fig. 64 Multichannel oscilloscope and rfft, example

Explanation

1. A composite signal is created by adding four signals with different frequencies, which are generated by ibaLogic's generators. The four single singnals are connected to a "probe" of the multichannel oscilloscope. The units of the X-axes (xUnit) have the default value 1.

2. The composite signal is the input of the function block "*FB_Collect_1024*" which had been created with Structured Text. The purpose of this function block is to transform the time-continuous input signal into an output which is an one-dimensional array with 1024 cells (*Time_Signal_1024*). In the same time the xUnits for time- and frequency-axes are evaluated.
   Refer to the box **Dynamic scaling** below.
   The latter values are connected with the inputs xUnit1 and xUnit2 of another "probe" of the multichannel oscilloscope (*FFT_1024)*.
   Finally, the function block generates a boolean trigger signal which is set TRUE for one cycle, whenever the the array has been filled (every 1024 cycles).

3

3 The signal *Time_Signal_1024* is then connected to the input of a rfft function block. Everytime the trigger signal is TRUE the rfft takes in the array which contains the amplitude values of the composite signal (1024 samples). By means of the FFT function the frequency spectrum of the composite signal is evaluated and written the output which is an array again but consisting of 512 cells and containing the frequency amplitudes. Each cell (index) of the output array corresponds to one frequency. The first cell (index = 0) corresponds to the constant comonent of the input signal (f = 0 Hz). Every following index corresponds to a higher frequency which is equal to the one before incremented by *xUnit_FreqSignal*. If, for example, *xUnit_FreqSignal = 0.0978* it's possible to describe a frequency range from 0 to 50 Hz (511 * 0.0978).

4 The function block *fb_const_compo_1024* eliminates the constant component by writing 0.0 into the first cell of the array resulting from the FFT, everytime the trigger is TRUE.

The display of the multichannel oscilloscope (probe *FFT_1024*) looks as follows:

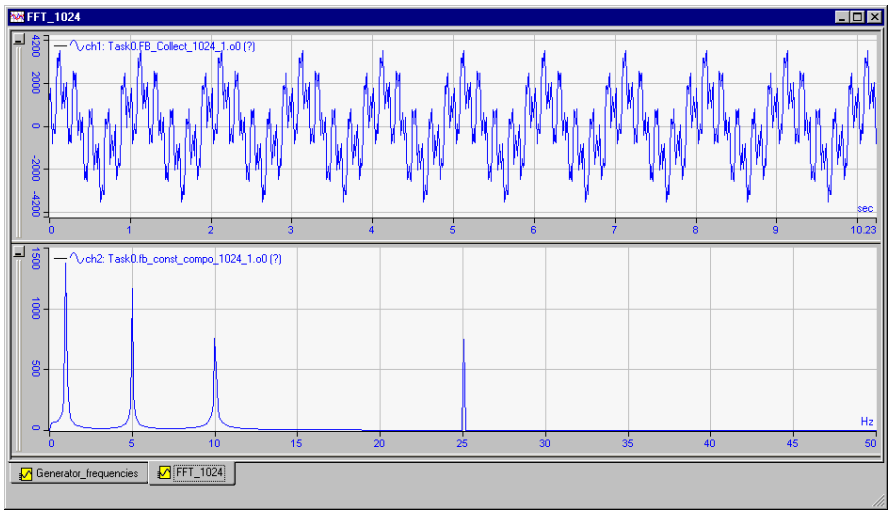


Fig. 65 Multichannel oscilloscope and rfft, result view

The upper strip shows a time-based graph of the composite signal consisting of 1024 samples.

The lower strip shows the resulting FFT graph which shows significant peaks at frequencies 1 Hz, 5 Hz, 10 Hz and 25 Hz which correspond exactly to the four generator frequencies.

Dynamic scaling

The XUnit of the time axis in the example above corresponds to the task cycle time (10 ms = 0.01 s). The XUnit of the frequency axis for the FFT-representation is the result of the computation of number of samples and time distance between the samples (XUnit time), considering the sampling theorema.

$$xUnit\_FreqSignal = \frac{1}{(xUnit\_TimeSignal * 2)(1024/2)}$$

In the multichannel oscilloscope the XUnit of the frequency axis is the scale index of one sample in the FFT-result array (output of the rfft function block), i.e. the distance on scale (in Hz) between two FFT-results.

IbaLogic should run in signal manager mode for a proper FFT-calculation.

3.14 Save the project against unintended changes

In the menu bar you'll find a button with the key-icon  . This command is a mean to lock the online layer, i.e. to prevent modifications of the project. It is still possible to navigate through the function block diagram and to open macro blocks in order to view.

If the key-button is pressed, then ...

- ❑ all editing functions are switched off
- ❑ the usual Windows functions, such as collapse, expand or close windows are disabled.
- ❑ to save, to read or to exit a program is not possible.
- ❑ the hardware settings are read-only.

If the key-button is pressed again, the editing function is unlocked again (see also next section).

3.15 Password protection and other protecting measures

A project can be protected by a password. If the password protection mode is enabled, a password is required in order to lock and unlock the layer with the key-button  .

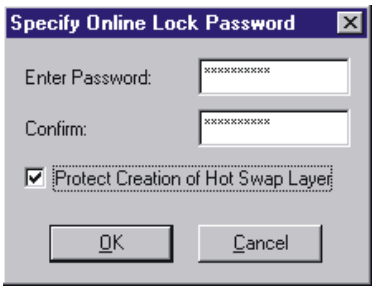


Fig. 66 Activate password protection

If the checkbox "*Protect Creation of Hot Swap Layer*" is enabled a Hot-Swap-Layer can not be opened, too.

3.16 The Hot-Swap layer

One of ibaLogic's unique features is the occasion to modify a layout during online operation without affecting the process. The modifications are to be applied (swapped) later whenever it is suitable. This is made possible by the use of a so called Hot-Swap layer, which is in fact a kind of workbench, independent from the running layout. Particularly modifications which would break up an existing network or which would shortly delete connection lines, e.g. when inserting a new function block between two others. The Hot-Swap-Layer is a crucial feature particularly for applications in continuous processing lines, like in paper production.

If created over the menu ↪ *HotSwap* ↪ *Create* or the command button  the Hot-Swap layer is an exact copy of the online program. While the online layer is indicated by a purple background color, the Hot-Swap layer is grey.

The Hot-Swap layer can be modidfied and tested (evaluated) like a usual layer. But the Hot-Swap layer won't be set online, i.e. the modification will be compiled and evaluated but it won't affect the outputs. Of course, the real process inputs are used for evaluation.

The Hot-Swap-Layer will not become the online layer until it is activated by the user with menu ↪ *HotSwap* ↪ *Apply to Online Layer.* The switch-over will be done smooth and correctly in terms of cycle and evaluation chronology.

Fig. 67 Create Hot-Swap layer

At any time it is possible to switch back and forth between Hot-Swap and Online layer by pressing the button  .

The menu command ↪ *HotSwap* ↪ *Close* will dispose all modifications if not stored as suggested.

3.16.1. Conception of data handling and memory in Hot-Swap

When the Hot-Swap layer is active, it has to be ensured that no operation via OPC gets lost. Also, locally stored information of function blocks have to be kept in memory during switch-over.

The method of ibaLogic ensures this by adding only the information of the new function blocks to the online layer while keeping the other information unchanged.

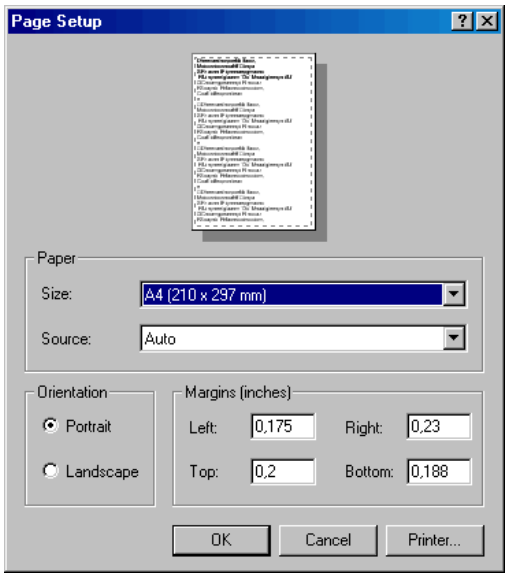
OPC input and output connectors (OTC) will not be evaluated in the Hot-Swap layer.

3.17 Printing a project

ibaLogic offers a variety of printer control functions which can be preset. Generally, the WYSIWYG-method (What You See Is What You Get) applies.

3.17.1. Setting the page size for a project

Basically, size and orientation of printed pages should be adjusted at the beginning of engineering a project. This is to avoid changes of print format in the future, and thus, additional work.



By using the menu ↪ *File* ↪ *Page Setup...* the window as shown on left side will open.

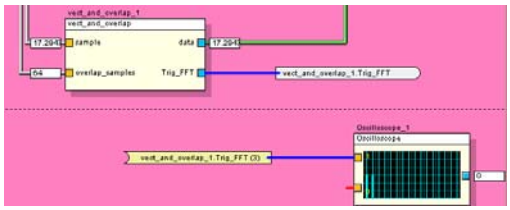
This is the place to make the print settings for the entire project.

It is recommended to use format A4 landscape or larger. Other formats, e.g. letter, are also available as templates.

According to these settings, the pages are marked in the function block diagram by a dotted line.

The margins are either I/O resource margins (for pages at the far right or at the far left) or dotted lines in case to devide two pages horizontally or vertically.

Never place a function block on a borderline because it could be cut when printed!



Left, you see such a borderline between two pages.

In order to keep the printout clear it is recommended to use IntraPage-connectors (IPC) if vertical connection lines cross a page border. With IPCs it's easier to track a signal. With version 3.80 of ibaLogic IPCs are created automatically when drawing a vertical connection line over a page border.

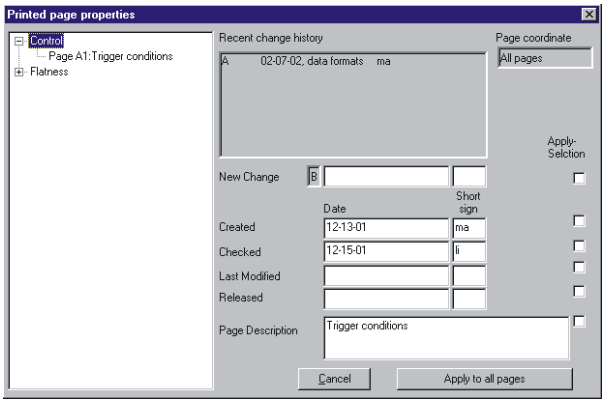
(see also chapter 3.9.2)

3.17.2. Inscription and layout of pages

Inscription and layout of the printed pages are designed in compliance with international standards in order to meet the usual requirements from technical documentation.

Every printed page shows references to creator and date of creation (both applied automatically at first page creation, taken from the general file settings), change notes and page description (title). These properties can be set and edited by using menu ↪ *Edit* ↪ *Page* ↪ *Page properties* or by a right mouseclick on an empty place in the page (↪ Edit menu)

The following dialog window will open:



The example (left) shows the tree structure of the layout.
The field in the upper middle part of the window shows the change history. Change notes have to be entered in the input field below. Date of creation and initials of the creator are written automatically at the time of page generation but they can be entered manually as well.

The short sign of the creator will be taken by default from the settings, made unter menu ↪*File* ↪*Settings* ↪*Edit Settings*.

In the lower field the page description (title) should be entered.

The coordinates of the page are indicated in the upper right corner.

When the data input is completed press the "Apply.."-button in order to save the inputs.

Note: Depending on the position of the highlighted bar in the tree structure, i.e. whether it marks a page or a task, the settings of the page properties apply to a single page or to all pages in the task. If only one page is selected, the check boxes "Apply Selection" are hidden. If a task is selected then the check boxes are available in order to control which information should be applied to all pages.

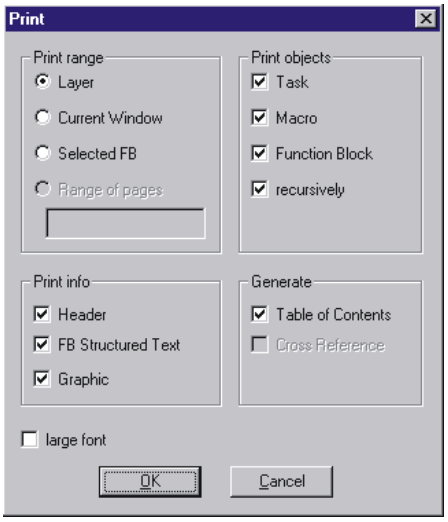
The page coordinates (i.e. page numbers) refer to the following matrix:

Letters refer to rows and numbers refer to columns.

A1	A2	...
B1	B2	...
...	...	...

3.17.3. Printer control settings

The subject of printing can be specified in order to avoid a waste of time and material.



A printout can cover an entire layout or parts of it. (Print range)

Layer refers to the entire Layout (all tasks).

Current Window refers to the current selected task.

Selected FB refers to a FB or group of FBs which are selected.

Furthermore, there is a selection of objects to be printed. The option "*recursively*" e.g. will lead to a printout of the contents of every macrotype which is used in the layout, once per task.

Additional information can be added to the printout. The option "*FB Structured Text*" will lead to a printout of the code of function blocks which are written in Structured Text.

"*Large font*" will promt the printer to use a larger font. It depends on the paper size which font is the better choice.

The option "*Table of contents*" will always add a coverpage and a table of contents with reference to the selected range and objects.

Please refer to the following table for some examples of frequent printout requests.

Printing....	Layer	Current Window	Selected FB	Task	Macro	Function Block	recursively	Header	FB ST	Graphic
complete layout with all tasks, Task parameters, Task as ST and graphical FBD but no internal macro information and no ST code of FBs	❑			❑				❑	❑	❑
as above but with internal macro information (graphic)	❑			❑	❑		❑	❑	❑	❑
as above and addtionally ST code of local FBs	❑			❑	❑	❑	❑	❑	❑	❑
only the graphical FBD of the currently selected task but no internal macro information		❑		❑						❑
only parameters of selected FBs but no ST code			❑			❑				

Table 8 Combinations of print settings

3.17.4. Adding your corporate logo on the printed pages

A specific graphic or picture can be placed on the cover page of a layout print. The corporate logo will appear on every page in the footer. With its first start, ibaLogic generates the files ibaLogo.bmp and ibaTitle.jpg and stores them in the folder ...\configuration. This is done even when the files are not there.

If you wish to include your own corporate logo (*ibaLogo.bmp*) and / or cover picture (*ibaTitle.jpg*) just replace these files by those with your logo, resp. picture, but using the same name and the same format.

The scaling of the logo, resp. picture, is done automatically according to the available space when printing.

3.17.5. Adding your corporate copyright note

A specific copyright note can be placed on every page. Like for the logo or the picture, you have to modify a standard file which is named *ibaCopy.txt*. This file is also stored in the folder *\configuration* and it may contain any text. The default setting is "Copyright © 2001". You may use any ASCII-editor to edit this text.

3.17.6. Printed pages

The following picture shows a typical page of a layout print with its special fields and their source of information.

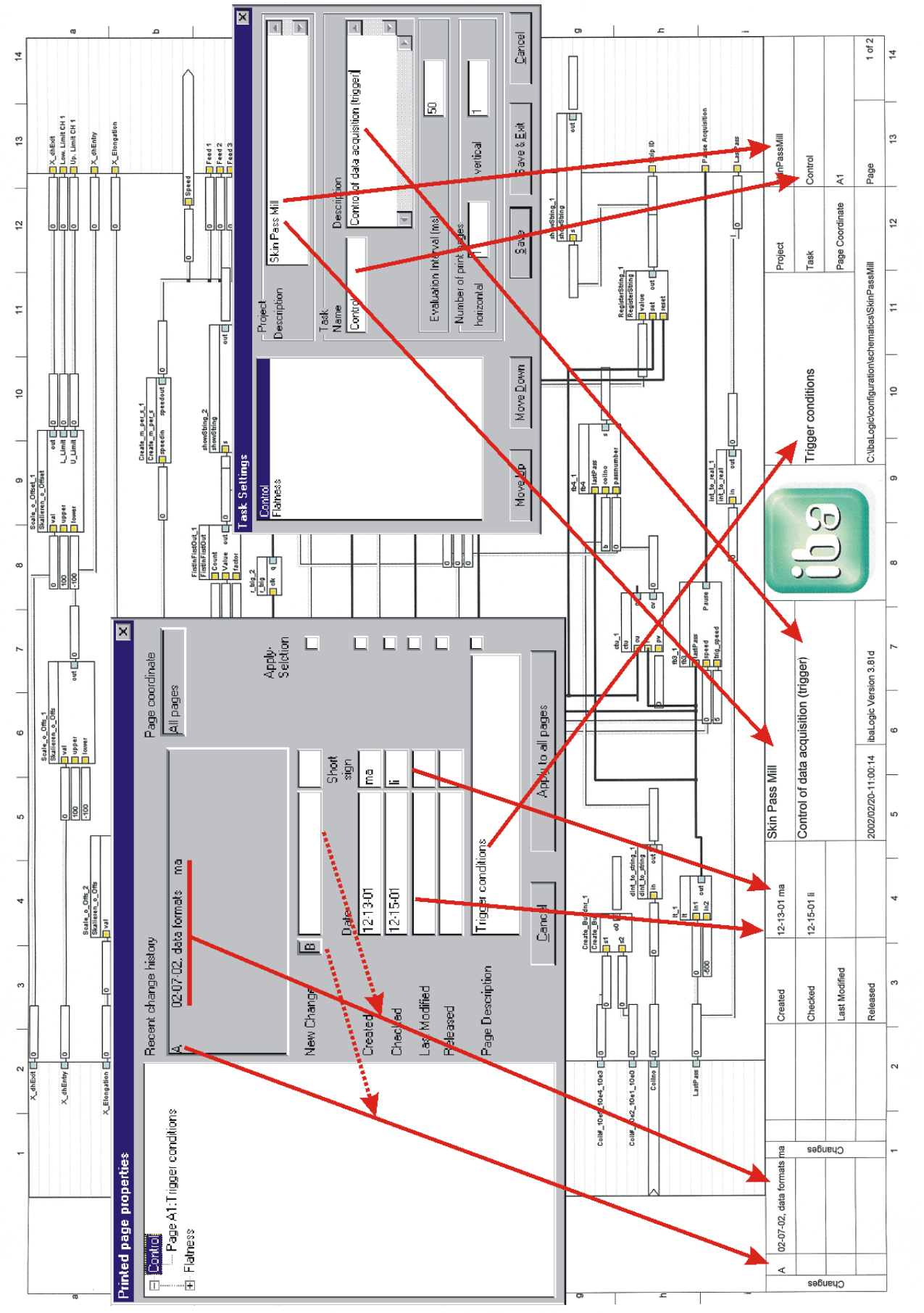
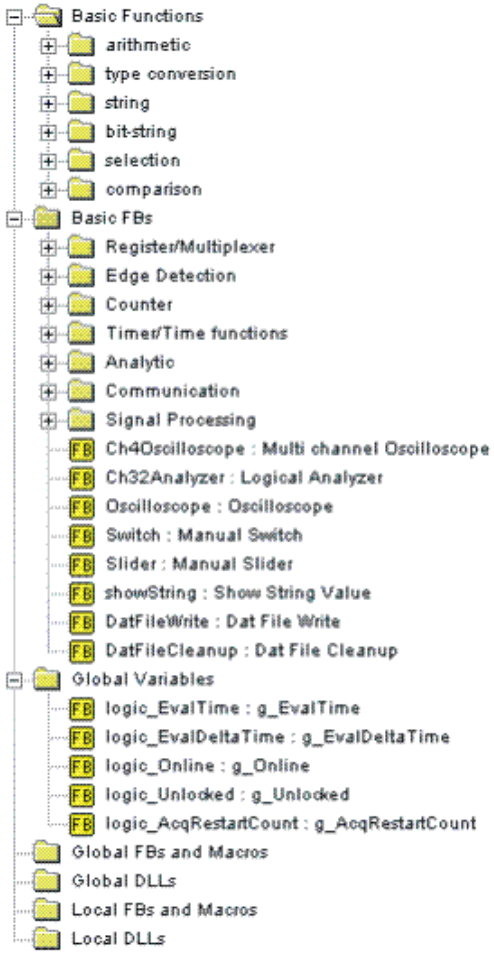


Fig. 68 Printed page, example

4 Functions and function blocks

ibaLogic's functions and function blocks are arranged in seven main groups in order to keep it clear and easy to handle. To get to the functions list as shown below, just click the „Functions"-tab in the resource area.



ibaLogic function block library

The seven main groups of functions and function blocks are:

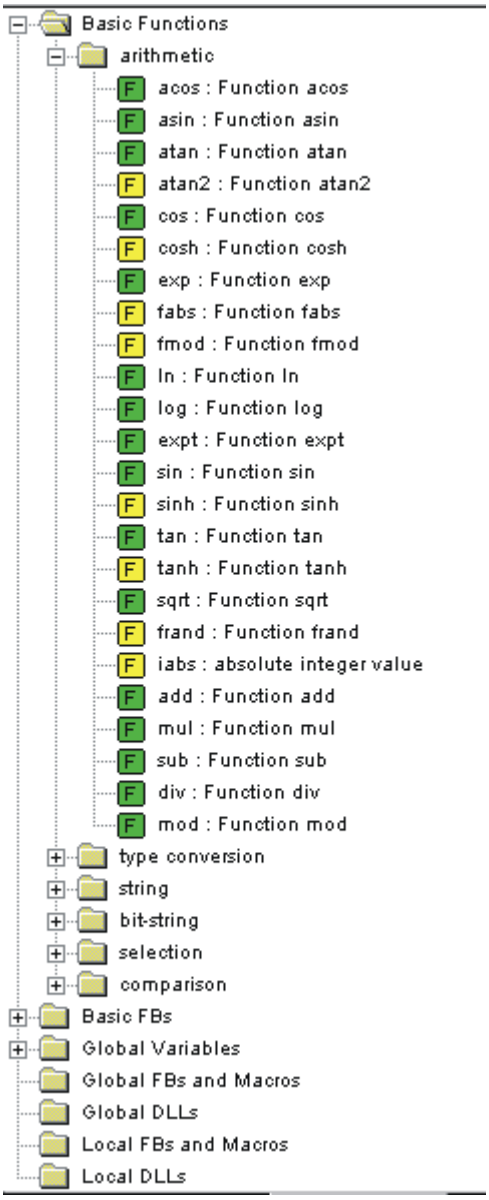
- Basic Functions
- Basic FBs
- Global Variables
- Global FBs and Macros
- Global DLLs
- Local FBs and Macros
- Local DLLs

Functions and function blocks which are recommended in accordance to the IEC 61131-3 standard are marked with a green icon:

Additional functions and function blocks which are provided by ibaLogic because they are useful and helpful are marked with a yellow icon:

4.1 Basic functions

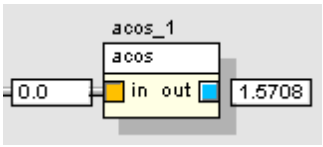
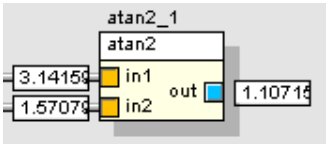
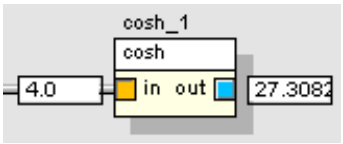
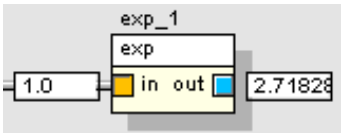
4.1.1. Arithmetic functions



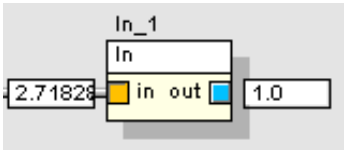
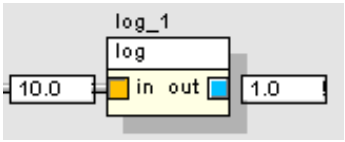
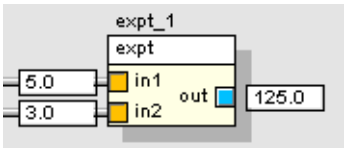
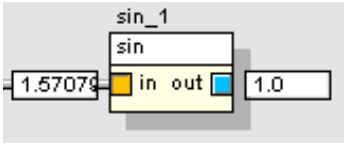
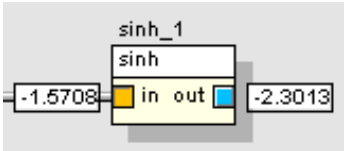
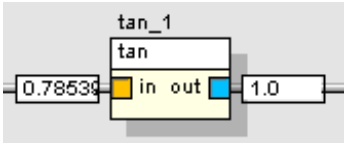
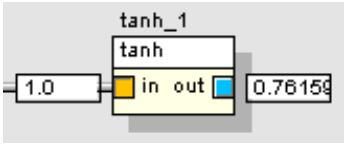
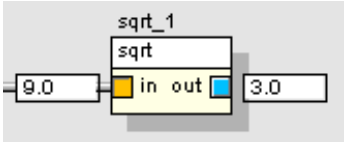
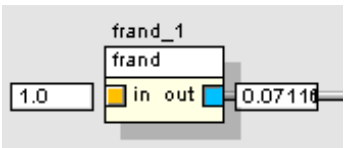
Overview of "Basic Functions", "arithmetic"

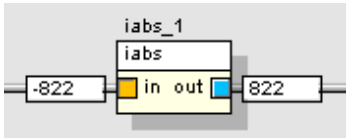
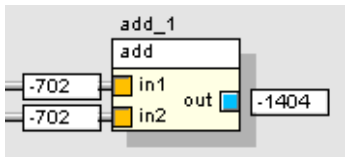
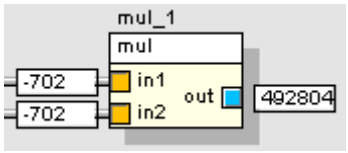
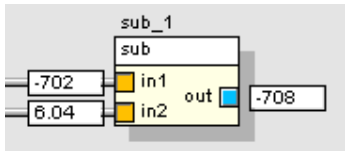
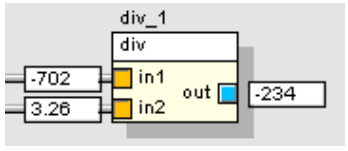
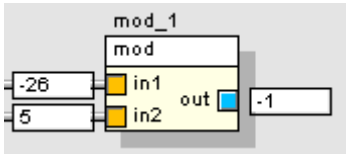
- acos cosine arc of arg
- asin sine arc of arg
- atan tangent arc of arg
- atan2 tangent arc of arg1 over arg2
- cos cosine of arg
- cosh cosine hyperbolic of arg
- exp natural exponential e** of arg
- fabs absolute value of arg (REAL)
- fmod floating point remainder of arg1 over arg2 (REAL)
- ln natural logarithm of arg
- log logarithm base 10 of arg
- expt exponentiation arg1 ** arg2
- sin sine of arg
- sinh sine hyperbolic of arg
- tan tangent of arg
- tanh tangent hyperbolic of arg
- sqrt square root of arg
- frand pseudo random number in the range from 0 to arg
- iabs absolute value of arg (INTEGER)
- add addition of arg1 plus arg2
- mul multiplication of arg1 by arg2
- sub subtraction of arg1 minus arg2
- div division of arg1 by arg2
- mod remainder of devision arg1/arg2

(arg = argument)

No.	Source Type	Arithmetic Functions Symbol	Target Type	Description, Example
1	LREAL		LREAL	**acos**: cosine arc of < arg > `Result:= acos(arg);` `Examples: 1.57079= acos(0.0);` `3.14159= acos(-1.0);`
2	LREAL		LREAL	**asin**: sine arc of < arg > `Result:= asin(arg);` `Examples: -1.57079= asin(-1.0);` `1.57079= asin(+1.0);`
3	LREAL		LREAL	**atan**: tangent arc of < arg > `Result:= atan(arg);` `Examples: 1.0000= atan(π/2.0);` `1.2626= atan(π);`
4	LREAL LREAL		LREAL	**atan2**: tangent arc of < arg1 > over <arg2> `Result:= atan2(arg1,arg2);` `Examples: 1.1071= atan2(π,π/2.0);`
5	LREAL		LREAL	**cos**: cosine of < arg > `Result:= cos(arg);` `Examples: -1.0000= cos(π);` `+1.0000= cos(2.0*π);`
6	LREAL		LREAL	**cosh**: cosine hyperbolic of < arg > `Result:= cosh(arg);` `Examples: +27.3082= cosh(4.0);` `+201.7156= cosh(-6.0);`
7	LREAL		LREAL	**exp**: natural exponential of < arg > `Result:= exp(arg);` `Examples: +2.71828= exp(1.0);` `+0.13533= exp(-2.0);`
8	LREAL		LREAL	**fabs**: absolute value of < arg > `Result:= fabs(arg);` `Examples: +4.06= fabs(-4.06);` `+3.89= fabs(+3.89);`
9	LREAL LREAL		LREAL	**fmod**: floating point remainder of < arg1 > over <arg2> `Result:= fmod(arg);` `Examples:` `+1.6789=fmod(5.6789,2.0);` `+1.862= fmod(+3.862,2.0);`

4

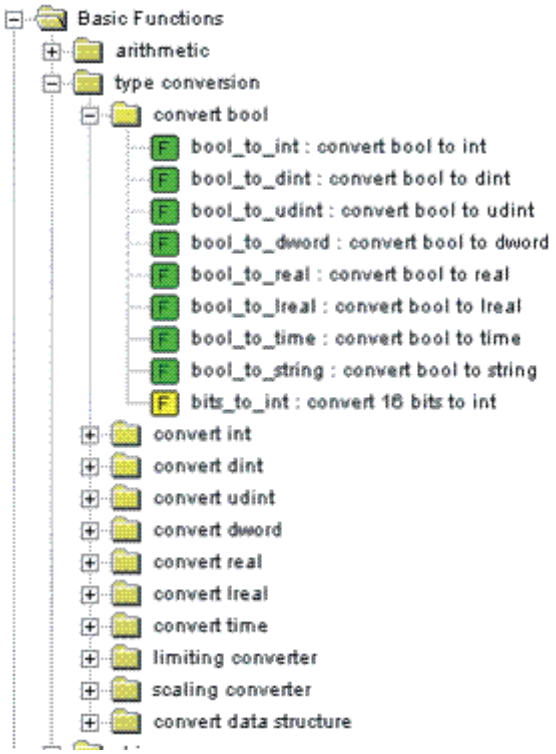
No.	Source Type	Arithmetic Functions Symbol	Target Type	Description, Example
10	LREAL		LREAL	**ln**: natural logarithm < arg > `Result:= ln(arg);` `Examples: +1.00= ln(2.71828);` `-4.00= ln(0.01831);`
11	LREAL		LREAL	**log**: Logarithm base 10 of < arg > `Result:= log(arg);` `Examples: +1.00= log(10.0);` `-4.00= log(0.0001);`
12	LREAL any no.		LREAL	**expt**: exponentiation < arg1 >**<arg2> `Result:= expt(arg1;arg2);` `Examples: +125.0= expt(5.0,3.0);` `+4.00= expt(16.0,0.5);`
13	LREAL		LREAL	**sin**: sine of < arg > `Result:= sin(arg);` `Examples: +1.00= sin(π/2.0);` `-0.8414= sin(-1.00);`
14	LREAL		LREAL	**sinh**: sine hyperbolic of < arg > `Result:= sinh(arg);` `Examples: -2.3013= sinh(-π/2.0);` `+2.3013= sinh(+π/2.0);`
15	LREAL		LREAL	**tan**: tangent arc of < arg > `Result:= tan(arg);` `Examples: +1.00= tan(+π/4.0);` `-1.00= tan(-π/4.0);`
16	LREAL		LREAL	**tanh**: tangent hyperbolic of < arg > `Result:= tanh(arg);` `Examples: +0.76159= tanh(1.00);` `-0.99627= tanh(-π);`
17	LREAL		LREAL	**sqrt**: square root of < arg > `Result:= sqrt(arg);` `Examples: +3.00= sqrt(9.00);` `+1.4142= sqrt(2.00);`
18	LREAL		LREAL	**frand**: generates a pseudo random number in the range from 0 to < arg > `Result:= frand(arg);` `Examples: +0.07116= frand(1.00);` `+0.92457= frand(1.00);`

No.	Source Type	Arithmetic Functions Symbol	Target Type	Description, Example
19	DINT/INT		INT/DINT	**iabs**: Absolute value of < arg > (INT / DINT) `Result:= iabs(arg);` `Examples: +822= iabs(-822);` `+342= iabs(+342);`
20	any no. any no.		any no.	**add**: addition of arguments arg1 + arg2 `Result:= add(arg1,arg2);` `Examples: -1404= add(-702,-702);` `+5.27= add(5.00,0.27);`
21	any no. any no.		any no.	**mul**: multiplication of arguments arg1 * arg2 `Result:= mul(arg1,arg2);` `Examples: 492804= mul(-702,-702);` `+1.350= mul(5.00,0.27);`
22	any no. any no.		any no.	**sub**: subtraction of arguments arg1 - arg2 `Result:= sub(arg1,arg2);` `Examples: -708= sub(-702,6.04);` `+4.73= sub(5.00,0.27);`
23	any no. any no.		any no.	**div**: division of arguments arg1/arg2 `Result:= div(arg1,arg2);` `Examples: -234= div(-702,3.26);` `+18.51= div(5.00,0.27);`
24	INT/DINT INT/DINT		INT/DINT	**mod**: remainder of division (Modulo) arg1/arg2 `Result:= mod(arg1,arg2);` `Examples: -1= mod(-26,5);` `+4= mod(326,7);`

Remark:
Functions in accordance with IEC are marked green, additional functions, provided by iba are marked yellow
LREAL FABS(ARG); absolute value of LREAL-numeral arg (IEC-functionname is "ABS")
DINT IABS(ARG); absolute value of DINT-numeral (IEC-functionname ist "ABS")

4

4.1.2. Type conversion



Overview "Basic Functions, Type Conversion"
There are the following groups of converting function blocks available:

- Convert BOOL
- Convert INT
- Convert DINT
- Convert UDINT
- Convert DWORD
- Convert REAL
- Convert LREAL
- Convert TIME
- Limiting converter
- Convert data structure

4.1.2.1. Rules for conversion

If variables and function blocks are to be connected in the program, ibaLogic checks the compatibility of data types automatically. If different data types are involved, usually a converter function is required and ibaLogic offers to enter one. Furthermore, the following rules for conversion apply:

- ❑ All signed integer operations are computed with **32-bit DINT** accuracy.
- ❑ If required, non-STRING values will be automatically converted into STRING, except data of type ARRAY.
- ❑ Standard functions are used for conversion. The name of the function derives from **<source type>_to_<target type>** (see examples).
- ❑ For target type BOOL the input value is coverted in "FALSE" if the input value is 0, 0.0, 16#0 or T#0ms. Else it's converted in "TRUE".
- ❑ DINT, UDINT and DWORD conversions are created with a copy of the current 4-byte date (32 bit).
- ❑ **REAL** to **DWORD** conversions are created with a copy of the current 4-byte date (32 bit).
- ❑ **LREAL** to **DWORD** conversions are created like REAL to DWORD, but LREAL is first converted in REAL.
- ❑ The conversion of **REAL/LREAL** in **DINT/UDINT** is done by a numeric computation, assuming that the permissible limits of value range are not violated.
- ❑ For data type "**TIME**" it is assumed that the input value "1" or "1.0" is given in the unit "second"
- ❑ A special function "**TRUNC**" converts LREAL to DINT without rounding.

- When a datatype of a large value range should be converted into a datatype with a smaller value range, a limiting converter is provided (offered) by the program automatically.

Table of conversions

	BOOL	INT	DINT	UDINT
BOOL	N/A	bool_to_int	bool_to_dint	bool_to_udint
INT	int_to_bool	N/A	int_to_dint	int_to_udint
DINT	dint_to_bool	dint_to_int limit_dint_to_int	N/A	dint_to_udint limit_dint_to_udint
UDINT	udint_to_bool	udint_to_int limit_udint_to_int	udint_to_dint limit_udint_to_dint	N/A
DWORD	dword_to_bool	dword_to_int	dword_to_dint	dword_to_udint
REAL	real_to_bool	real_to_int limit_real_to_int	real_to_dint limit_real_to_dint	real_to_udint limit_real_to_udint
LREAL	lreal_to_bool	lreal_to_int limit_lreal_to_int	lreal_to_dint limit_lreal_to_dint trunc	lreal_to_udint limit_lreal_to_udint
TIME	time_to_bool	time_to_int	time_to_dint	time_to_udint
STRING	N/A	N/A	(atoi)	N/A
Bits	N/A	bits_to_int	N/A	N/A

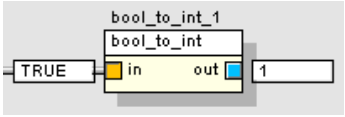
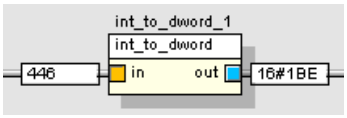
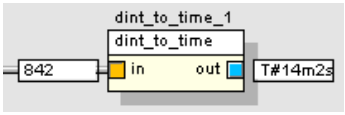
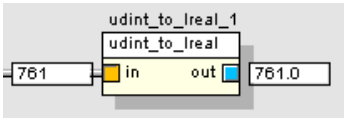
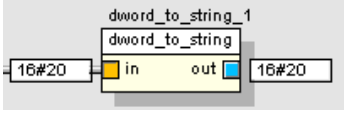
	DWORD	REAL	LREAL	TIME
BOOL	bool_to_dword	bool_to_real	bool_to_lreal	bool_to_time
INT	int_to_dword	int_to_real	int_to_lreal	int_to_time
DINT	dint_to_dword	dint_to_real	dint_to_lreal	dint_to_time
UDINT	udint_to_dword	udint_to_real	udint_to_lreal	udint_to_time
DWORD	N/A	dword_to_real	dword_to_lreal	dword_to_time
REAL	real_to_dword	N/A	real_to_lreal	real_to_time
LREAL	lreal_to_dword	lreal_to_real limit_lreal_to_real	N/A	lreal_to_time
TIME	time_to_dword	time_to_real	time_to_lreal	N/A
STRING	N/A	(atof, atofFmt)	N/A	N/A
Bits	N/A	N/A	N/A	N/A

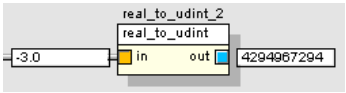
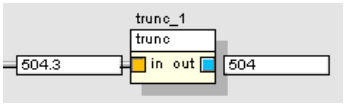
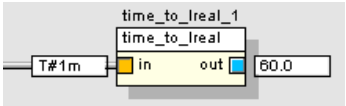
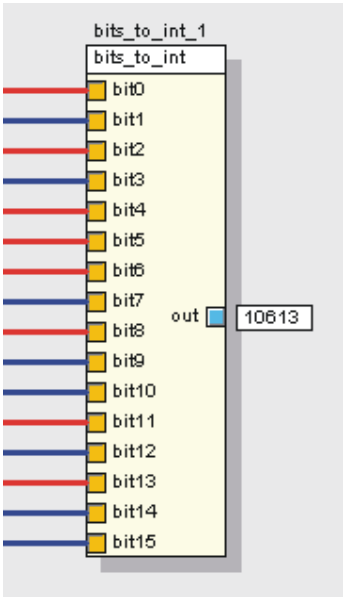
	STRING	bits	char
BOOL	bool_to_string	N/A	N/A
INT	int_to_string	int_to_bits	N/A
DINT	dint_to_string	N/A	N/A
UDINT	udint_to_string	N/A	N/A
DWORD	dword_to_string	N/A	dword_to_char
REAL	real_to_string	N/A	N/A
LREAL	lreal_to_string	N/A	N/A
TIME	time_to_string	N/A	N/A
STRING	N/A	N/A	N/A
Bits	N/A	N/A	N/A

For target type "**STRING**" the input values are converted as follows:

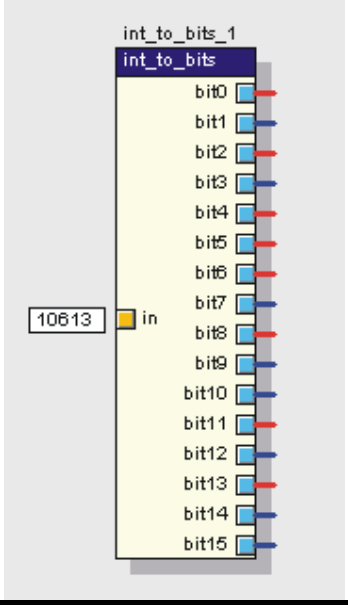
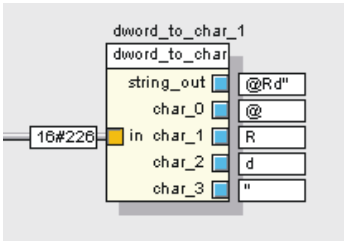
- "FALSE" or "TRUE", for source type BOOL
- Decimal row of characters (e.g. "-1234" or "123.456") for source types INT, DINT, UDINT, REAL or LREAL.
- Hex sequence of characters (e.g. "16#56AF3") for source type DWORD.
- Time sequence of characters (e.g. "T#5m35s200ms") for source type TIME

4.1.2.2. General type converting functions

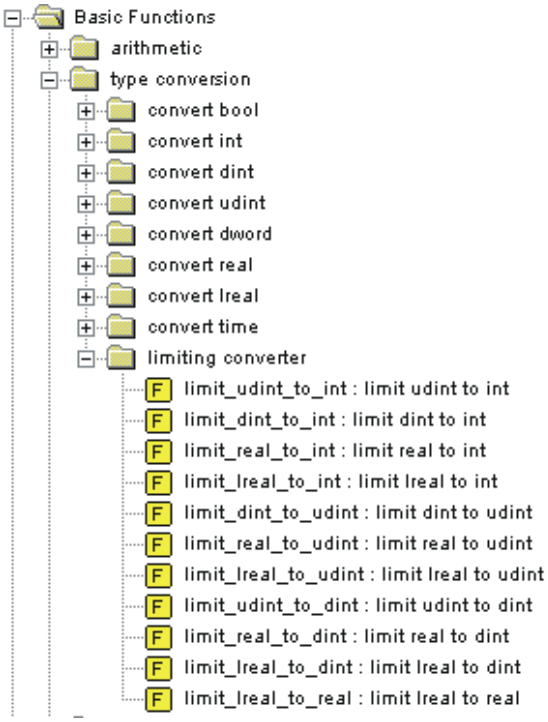
No.	Source Type	Type Conversion Symbol	Target Type	Description, Examples
1	BOOL	Example: bool_to_int 	INT DINT UDINT DWORD REAL LREAL TIME STRING	Convert BOOL bool_to_int: TRUE => 1; bool_to_dint: FALSE => 0; bool_to_udint: TRUE => 1; bool_to_dword: TRUE => 16#1; bool_to_real: TRUE => 1.0; bool_to_lreal: FALSE => 0.0; bool_to_time: TRUE => T#1s; bool_to_string: TRUE => TRUE;
2	INT	Example: int_to_dword 	BOOL DINT UDINT DWORD REAL LREAL TIME STRING	Convert INTEGER int_to_bool: 446 => TRUE; int_to_dint: 446 => 446; int_to_udint: 446 => 446; int_to_dword: 446 => 16#1BE; int_to_real: 446 => 446.0; int_to_lreal: 446 => 446.0; int_to_time: 446 => T#7m26s; int_to_string: 446 => 446;
3	DINT	Example: dint_to_time 	BOOL INT UDINT DWORD REAL LREAL TIME STRING	Convert Double INTEGER dint_to_bool: 842 => TRUE; dint_to_int: 842 => 842; dint_to_udint: 842 => 842; dint_to_dword: 842 => 16#34A; dint_to_real: 842 => 842.0; dint_to_lreal: 842 => 842.0; dint_to_time: 842 => T#14m2s; dint_to_string: 842 => 842;
4	UDINT	Example: udint_to_lreal 	BOOL INT DINT DWORD REAL LREAL TIME STRING	Convert Unsigned Double INTEGER udint_to_bool: 761 => TRUE; udint_to_int: 761 => 761; udint_to_dint: 761 => 761; udint_to_dword: 761 => 16#2F9; udint_to_real: 761 => 761.0; udint_to_lreal: 761 => 761.0; udint_to_time: 761 => T#12m41s; udint_to_string: 761 => 761;
5	DWORD	Example: dword_to_string 	BOOL INT DINT UDINT REAL LREAL TIME STRING	Convert Double WORD dword_to_bool: 16#20 => TRUE; dword_to_int: 16#20 => 32; dword_to_dint: 16#20 => 32; dword_to_udint: 16#20 => 32; dword_to_real: 16#20 => 4.48416E-04; dword_to_lreal: 16#20 => 4.48416E-04; dword_to_time: 16#20 => T#3.2ms; dword_to_string: 16#20 => 16#20;

No.	Source Type	Type Conversion Symbol	Target Type	Description, Examples
6	REAL	Example: real_to_udint 	BOOL INT DINT UDINT DWORD LREAL TIME STRING	Convert REAL real_to_bool: -3.0 => TRUE; real_to_int: -3.0 => -3; real_to_dint: -3.0 => -3; real_to_udint: -3.0 => 4294967294; real_to_dword: -3.0 => 16#C0400000; real_to_lreal: -3.0 => -3.0; real_to_time: -3.0 => T#-3s; real_to_string: -3.0 => -3.0;
7	LREAL	Example: trunc (LREAL zu DINT ohne Runden) 	BOOL INT DINT UDINT DWORD REAL TIME STRING TRUNC	Convert LREAL lreal_to_bool: 0.0 => FALSE; lreal_to_int: 504.3 => 504; lreal_to_dint: 1.6 => 2; lreal_to_udint: 504.3 => 504; lreal_to_dword: 504.3 => 16#43FC2666; lreal_to_real: 504.3 => 504.3; lreal_to_time: 504.3 => T#8m24s300ms; lreal_to_string: 504.3 => 504.3; trunc: 1.6 => 1;
8	TIME	Example: time_to_lreal 	BOOL INT DINT UDINT DWORD REAL LREAL STRING	Convert TIME time_to_bool: T#1m => TRUE; time_to_int: T#1m => 60; time_to_dint: T#-2s500ms => -3; time_to_udint: T#1s => 1; time_to_dword: T#1m => 16#927C0; time_to_real: T#1m => 60.0; time_to_lreal: T#10.5ms => 0.0104; time_to_string: T#1m => T#1m;
9	BOOL	Example: bits_to_int 	INT	Convert 16 bits to int bits_to_int: bit0 = TRUE bit1 = FALSE bit2 = TRUE bit3 = FALSE bit4 = TRUE bit5 = TRUE bit6 = TRUE bit7 = FALSE bit8 = TRUE bit9 = FALSE bit10 = FALSE bit11 = TRUE bit12 = FALSE bit13 = TRUE bit14 = FALSE bit15 = FALSE } = 10613

4

No.	Source Type	Type Conversion Symbol	Target Type	Description, Examples
10	INT	Example: int_to_bits 	BOOL	Convert int to 16 bits int_to_bits: 10613 = { bit0 = TRUE, bit1 = FALSE, bit2 = TRUE, bit3 = FALSE, bit4 = TRUE, bit5 = TRUE, bit6 = TRUE, bit7 = FALSE, bit8 = TRUE, bit9 = FALSE, bit10 = FALSE, bit11 = TRUE, bit12 = FALSE, bit13 = TRUE, bit14 = FALSE, bit15 = FALSE
11	DWORD	Example: dword_to_char 	STRING (4 chars)	Convert DWORD in 4 chars STRING dword_to_char: 16#22645240 = { @ R d "

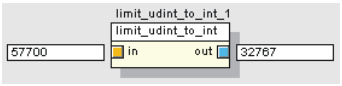
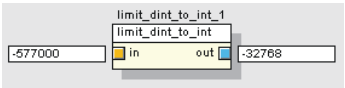
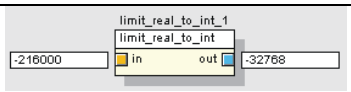
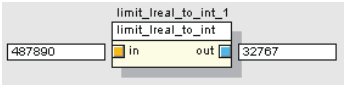
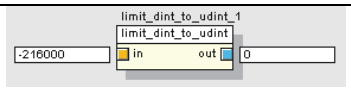
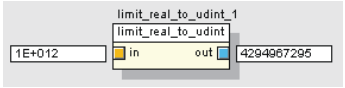
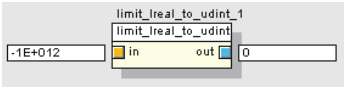
4.1.2.3. Limiting converters

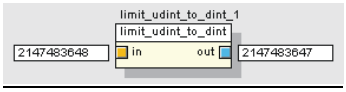
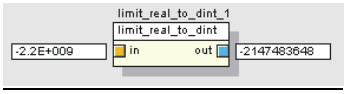
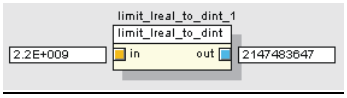
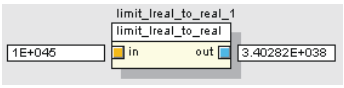


Overview "Limiting Converter"

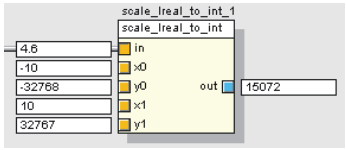
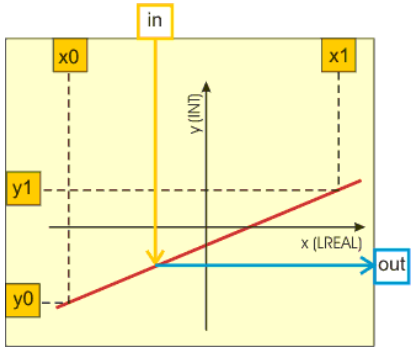
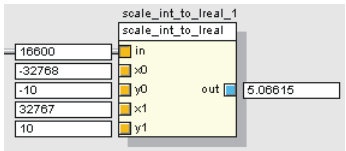
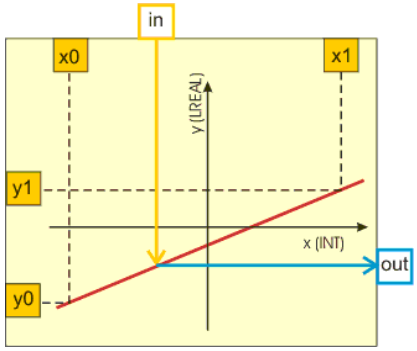
Limiting converters are function blocks of a special kind as they convert one data type into another and limit the output value to the max. / min. limits of the target data type if they are exceeded by the input value.

- Limit UDINT to INT
- Limit DINT to INT
- Limit REAL to INT
- Limit LREAL to INT
- Limit DINT to UDINT
- Limit REAL to UDINT
- Limit LREAL to UDINT
- Limit UDINT to DINT
- Limit REAL to DINT
- Limit LREAL to DINT
- Limit LREAL to REAL

No.	Source Type	Limiting Converter Symbol	Target Type	Description, Examples
1	UDINT		INT	Limit udint to int limit_udint_to_int: 577000 => 32767;
2	DINT		INT	Limit dint to int limit_dint_to_int: 577000 => 32767;
3	REAL		INT	Limit real to int limit_real_to_int: -216000 => -32768;
4	LREAL		INT	Limit lreal to int limit_lreal_to_int: -216000 => -32768;
5	DINT		UDINT	Limit dint to udint limit_dint_to_udint: -216000 => 0;
6	REAL		UDINT	Limit real to udint limit_real_to_udint: 1*E+12 => 4294967295;
7	LREAL		UDINT	Limit lreal to udint limit_lreal_to_udint: -1*E+12 => 0;

No.	Source Type	Limiting Converter Symbol	Target Type	Description, Examples
8	UDINT		DINT	Limit udint to dint limit_udint_to_dint:2147483648 =>2147483647;
9	REAL		DINT	Limit real to dint limit_real_to_dint: -2.2*E+09 => -2147483648;
10	LREAL		DINT	Limit lreal to dint limit_lreal_to_dint: 2.2*E+09 => 2147483648;
11	LREAL		REAL	Limit lreal to real limit_lreal_to_real: 1*E+45 => 3.402823466*E+38

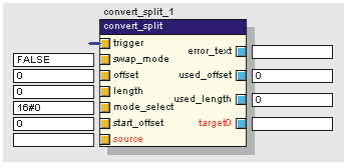
4.1.2.4. Scaling converters

No.	Source Type	Scaling Converters Symbol	Target Type	Description, Examples
1	LREAL LREAL INT LREAL INT		INT	**Scale_lreal_to_int** This function converts a LREAL value (e.g. a physical quantity) into an INTEGER value (e.g. an analog output) using a linear scaling. scale_lreal_to_int: 4.6 => 15072 with –10.0 => -32768 and +10.0 => 32767  Implementation: `diff_x := x1 - x0;` `if ( diff_x <> 0.0 )` `then` `  a := ( y1 – y0 ) / diff_x;` `  b := y0 - a * x0;` `  dout := a * in + b;` `  out := limit_lreal_to_int( dout );` `else` `  set_valid( out, FALSE );` `end_if;`
2	INT INT LREAL INT LREAL		LREAL	**Scale_int_to_lreal** This function converts an INTEGER value (e.g. an analog output) into a LREAL value (e.g. a physical quantity) using a linear scaling. scale_int_to_lreal: 16600 => 5.06615 mit -32768 =>–10.0 und 32767 => +10.0  Implementation: `diff_x := x1 - x0;` `if ( diff_x <> 0.0 )` `then` `  a := ( y1 - y0 ) / diff_x;` `  b := y0 - a * x0;` `  out = a * in + b;` `else` `  set_valid( out, FALSE );` `end_if;`

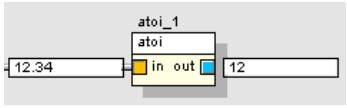
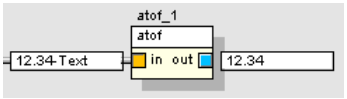
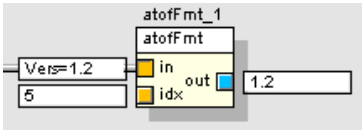
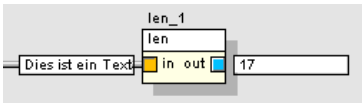
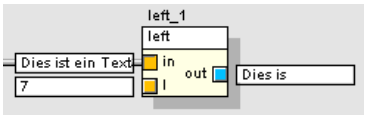
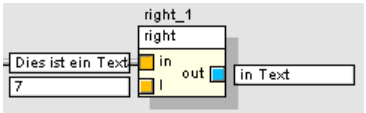
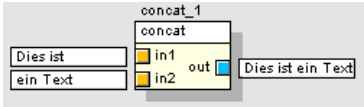
4.1.2.5. Convert data structure

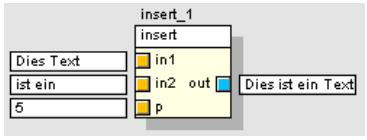
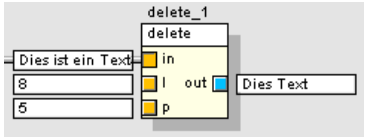
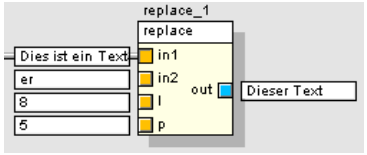
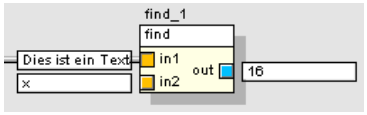
This function is used for exchange of data structures with external systems which use more complex data structures than ibaLogic.

No.	Source Type	Data Structure Converters Symbol	Target Type	Description, Examples
1	BOOL ARRAY ARRAY ARRAY ARRAY UDINT UNTYPED		STRING ARRAY ARRAY UNTYPED	**Convert_collect** This function is used for collecting several data elements of various or same types (sourceX) and putting them together in one mutual data structure (target). Up to 58 inputs (source 0...57) can be processed. Each source input is overloadable, i.e. different data types may be connected, including 4-dimensional arrays. Input parameters: *trigger* (BOOL): The function will only be evaluated if trigger = TRUE. *swap_mode* (Array of BOOL): If a bit of this array is TRUE, the data element at the corresponding source input will be swapped (depending on target system). *offset* (Array of UDINT): Byte offset per source in the target structure (target); if = 0 the data element will be written right behind the previous one. *length* (Array of UDINT): Byte length per source in the target structure; if = 0 the maximum length will be used. *mode_select* (Array of DWORD): not used. The index of these arrays (0...57) is assigned to the source inputs. *start_offset* (UDINT): Start address in the target structure, where the entries of the source data should begin. *sourceX* (untyped): Input data (X = 0...57) Output parameters: *error_text* (STRING): Status message *used_offset* (Array of UDINT): Used offset per source *used_length* (Array of UDINT): Used length per source *target* (untyped): Target data structure

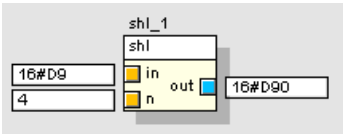
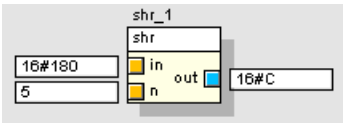
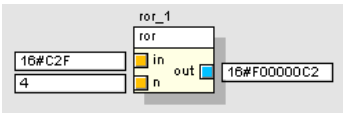
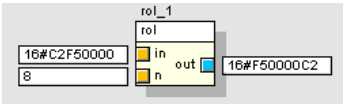
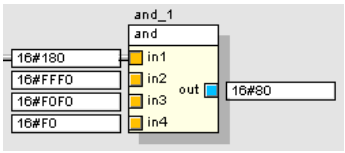
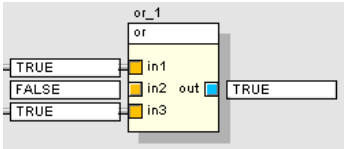
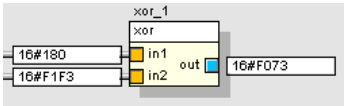
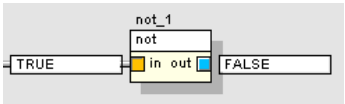
No.	Source Type	Data Structure Converters Symbol	Target Type	Description, Examples
2	BOOL ARRAY ARRAY ARRAY ARRAY UDINT UNTYPED		STRING ARRAY ARRAY UNTYPED	**Convert_split** This function provides the inverse function of Convert_collect and works accordingly. An input data structure (source) can be split up into various data elements of different or same types. For dismantling and reading the data structure correctly the position, length and type of the included data elements must be known. Input parameters: *trigger* (BOOL): The function will only be evaluated if trigger = TRUE. *swap_mode* (ARRAY of BOOL): If a bit of this array is TRUE, the corresponding data element will be swapped (depending on source system). *offset* (Array of UDINT): Byte offset per target in the source structure (source); if = 0 the data element can be found right behind the prvious one. *length* (Array of UDINT): Byte length per target in the source structure; if = 0 the maximum length will be read. *mode_select* (Array of DWORD): not used The index of these arrays (0...57) is assigned to the target outputs. *start_offset* (UDINT): Start address in the source structure, where the above mentioned target data had been entered. *source* (untyped): Input data structure Output parameters: *error_text* (STRING): Status message *used_offset* (Array of UDINT): Used offset per target *used_length* (Array of UDINT): Used length per target *targetx* (untyped): Target data

4.1.3. String functions

No.	Source Type	String Functions Symbol	Target Type	Description, Examples
1	STRING		DINT	**atoi**: Converts STRING to INTEGER `Result:= atoi(string);` `Examples: 12= atoi('12.34');` `1234= atoi('1234-Text');`
2	STRING		REAL	**atof**: Converts STRING to REAL `Result:= atof(string);` `Examples: 12.34= atof('12.34');` `12.34= atof('12.34-Text');`
3	STRING DINT		REAL	**atofFmt**: Converts STRING to REAL, beginning at start index"idx" `Result:= atofFmt(string,idx);` `Examples:` `1.2= atofFmt('Vers=1.2',5);` `54.32= atofFmt('a:=54.32',3);`
4	UDINT		UTC-Time-String	**UtcTimeToString**: Converts a UTC-time constant to a time-STRING `Result:= UtcTimeToString(arg);` `Examples: '2001/08/10.11:05:49'= Utc-TimeToString(997441549);` `'1970/01/01.00:00:01'= UtcTimeTo-String(1);`
5	STRING		DINT	**len**: Length of a string `Result:= len(string);` `Examples:` `17= len('Dies ist ein Text');` `4= len('Text');`
6	STRING DINT		STRING	**left**: Left part of a string, of a length of "l" (chars) `Result:= left(string,l);` `Examples: 'Dies is'= left('Dies ist ein Text',7);` `'Die'= left('Dies ist ein Text',3);`
7	STRING DINT		STRING	**right**: Right part of a string, of a length of "l" (chars) `Result:= right(string,l);` `Examples: 'in Text'= right('Dies ist ein Text',7);` `'ext'= right('Dies ist ein Text',3);`
8	STRING DINT DINT		STRING	**mid**: Excerpt of a string of a length of "l" (chars) beginning at position "p" `Result:= mid(string,l,p);` `Examples: 'es is'= mid('Dies ist ein Text',5,3);` `'ein'= mid('Dies ist ein Text',3,10);`
9	STRING STRING		STRING	**concat**: concatenates two strings to one `Result:= concat(string1,string2);`

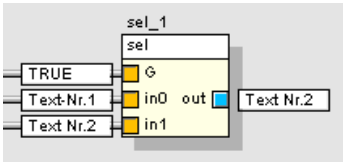
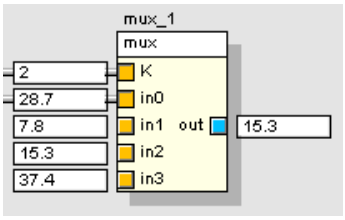
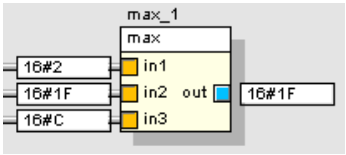
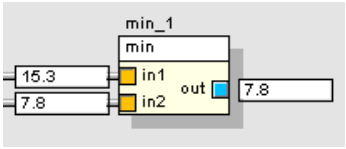
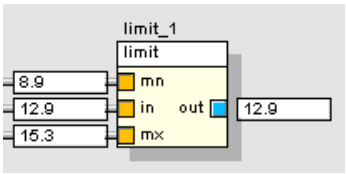
No.	Source Type	String Functions Symbol	Target Type	Description, Examples
				Examples: 'Dies ist ein Text'= concat ('Dies ist','ein Text'); 'ABCD'= concat('AB','CD');
10	STRING STRING DINT		STRING	**insert**: insert string "in2" in string "in1" at position "p" Result:=insert(string1,string2,p); Examples: 'Dies ist ein Text'= insert ('Dies Text','ist ein ',5); 'ABCDE'= insert ('AE','BCD', 1);
11	STRING DINT DINT		STRING	**delete**: delete "l" chars of a string, beginnig at position "p" Result:= delete(string,l,p); Examples: 'Dies Text'= delete('Dies ist ein Text',8,5); 'BCD'= delete('ABCDE',3,1);
12	STRING STRING DINT DINT		STRING	**replace**: replace "l" chars of string "in1" by "in2" beginning at position "p" Result:= replace(string1,string2,l,p); Examples: 'Dieser Text'= replace('Dies ist ein Text', 'er', 8, 5); 'ABXE'= replace('ABCDE','X', 2,3);
13	STRING STRING		DINT	**find**: find the first position where any char of string "in2" matches chars in string "in1" Result:= find(string1,string2); Examples: 16= find('Dies ist ein Text', 'x'); 1= find('Dies ist ein Text', 'exD');
Remark: Functions in accordance with IEC are marked green, additional functions, provided by iba are marked yellow Values of integer variables must not be signed negativ (<0)				

4.1.4. Bit-Shift functions and logical operations

No.	Source Type	Bit-Shift Functions Symbol	Target Type	Description, Examples
1	DWORD DINT		DWORD	**shl**: Left shift of "in" by "n" MOD 32 bits, zero-filled from right `Result:= shl(in,n);` `Examples: 16#D90= shl(16#D9,4);` `16#180= shl(16#C,5);`
2	DWORD DINT		DWORD	**shr**: Right shift of "in" by "n" MOD 32 bits, zero-filled from left `Result:= shr(in,n);` `Examples: 16#C= shr(16#180,5);` `16#D9= shr(16#D90,4);`
3	DWORD DINT		DWORD	**ror**: right rotation of "in" by "n" MOD 32 bits `Result:= ror(in,n);` `Examples:` `16#F00000C2= ror(16#C2F,4);` `16#F500000C= ror(16#CF5,8);`
4	DWORD DINT		DWORD	**rol**: left rotation of "in" by "n" MOD 32 bits `Result:= rol(in,n);` `Examples:` `16#F50000C2= rol (16#C2F50000,8);` `16#45678123= rol (16#12345678,12);`
5	Any bit ... ... Any bit		DWORD/ BOOL	**and**: Logical AND-operation of input variables (DWORD / BOOL) `Result:= and(in1,in2,...in_n);` `Examples:16#80= and(16#180, 16#FFF0,` `16#F0F0, 16#F0);` `FALSE= and(TRUE,FALSE,TRUE);`
6	Any bit ... Any bit		DWORD/ BOOL	**or**: Logical OR-operation of input variables (DWORD / BOOL) `Result:= or(in1,in2,...in_n);` `Examples:TRUE=or(TRUE,FALSE,TRUE);` `16#F1F3=or(16#180,16#F0F0,16#3);`
7	Any bit ... Any bit		DWORD/ BOOL	**xor**: Logical XOR-operation of input variables (DWORD / BOOL) `Result:= xor(in1,in2,...in_n);` `Examples: FALSE= xor(TRUE,TRUE);` `16#F073=xor(16#180,16#F13);`
8	Any bit		DWORD/ BOOL	**not**: Logical NOT-operation (negation) of input variable (DWORD / BOOL) `Result:= not(in);` `Examples: FALSE= not(TRUE);` `16#FFFFFE7F=not(16#180);`

Remark:
The number of inputs of function blocks "AND", "OR" and "XOR" is free to be altered. To alter the number of inputs double click on the function block and change the "In"-variable under I/O-connectors by entering the desired number oder clicking the arrows up / down.

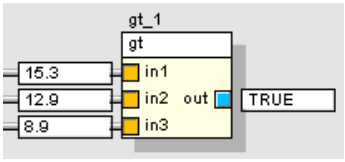
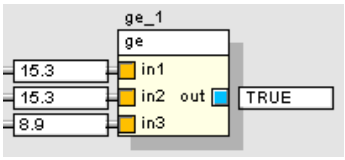
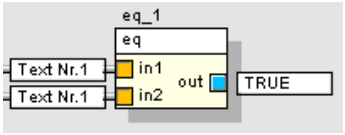
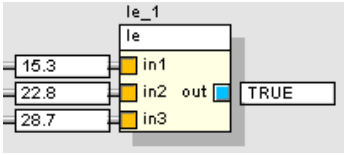
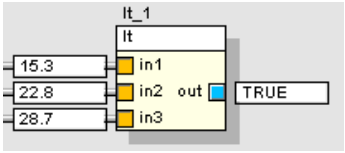
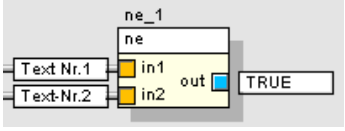
4.1.5. Selection- and MIN- / MAX-functions

No.	Source Type	Selection Functions Symbol	Target Type	Description, Examples
1	BOOL Any type Any type		Any type	**sel**: selection (1 out of 2) with binary switch "G" `out:= in0, if G = FALSE [0];` `out:= in1, if G = TRUE [1];` `Result:= sel(G,in0,in1);`
2	BOOL Any type .... Any type		Any type	**mux**: selection (1 out of n) by DINT-selector "K" `out:= in0, if K = 0;` `out:= in1, if K = 1;` `out:= in2, if K = 2;` `out:= inn, if K = n;` `out:= last value, if K >3;` `Result:= mux(K,in0,in1,in2,in3);`
3	Any type ..... Any type		Any type	**max**: maximum value of inputs (1..n) `Result:= max(in1,in2,inn);` `Examples:` `16#1F= max(16#2,16#1F,16#C);` `15.3 = max(12.3,7.8,15.3);`
4	Any type ..... Any type		Any type	**min**: minimum value of inputs (1..n) `Result:= min(in1,in2,inn);` `Examples:` `16#2= min(16#2,16#1F,16#C);` `7.8 = min(15.3,7.8);`
5	Any type Any type Any type		Any type	**limit**: linitation of input variable "in" between "mn" (minimum) and "mx" (maximum) `Result:= limit(in,mn,mx);` `Examples:` `12.9 = limit(12.9,8.9,15.3);` `15.3 = limit(17.6,8.9,15.3);` `8.9 = limit(2.0,8.9,15.3);`

Remark:
- "Any type": any elemental datatype BOOL/INT/DINT/UDINT/DWORD/REAL/LREAL/TIME/STRING
- The number of inputs of function blocks "mux", "max" and "min" is free to be altered. To alter the number of inputs double click on the function block and change the "In"-variable under I/O-connectors by entering the desired number oder clicking the arrows up / down.

4

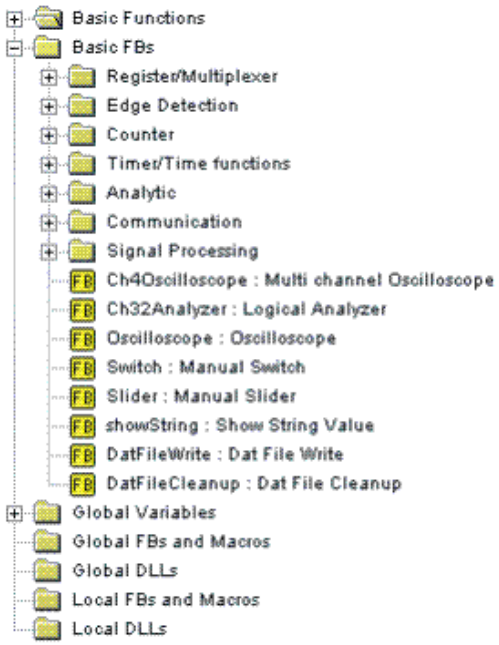
4.1.6. Comparison functions

No.	Source Type	Comparison Functions Symbol	Target Type	Description, Examples
1	Any type ..... Any type		BOOL	**gt**: "greater than" (1 out of n) `TRUE, if in1 > in2 > in3;` `FALSE, if in1 <= in2 <= in3` `Result:= gt(in1,in2,in3);` `Examples: TRUE= gt(15.3,12.9,8.9);` `FALSE= gt(15.3,6.8,8.9);`
2	Any type ..... Any type		BOOL	**ge**: "greater than or equal" (1 out of n) `TRUE, if in1 >= in2 >= in3;` `FALSE, if in1 < in2 < in3` `Result:= ge(in1,in2,in3);` `Examples: TRUE= ge(15.3,15.3,8.9);` `FALSE= ge(15.3,15.3,18.6);`
3	Any type Any type		BOOL	**eq**: "equal" (1 aus n) `TRUE, if in1 = in2 = in3;` `FALSE, if in1 <> in2 <> in3` `Result:= eq(in1,in2,in3);` `Examples:` `TRUE= eq('Text 1','Text 1');` `FALSE= eq(15.3,15.3,18.6);`
4	Any type ..... Any type		BOOL	**le**: "less than or equal" (1 out of n) `TRUE, if in1 <= in2 <= in3;` `FALSE, if in1 > in2 > in3` `Result:= le(in1,in2,in3);` `Examples: TRUE= le(15.3,22.8,28.7);` `FALSE= le(15.3,8.9,6.8);`
5	Any type ..... Any type		BOOL	**lt**: "less than" (1 out of n) `TRUE, if in1 < in2 < in3;` `FALSE, if in1 >= in2 >= in3` `Result:= lt(in1,in2,in3);` `Examples:` `TRUE= lt(15.3,22.8,28.7);` `FALSE= lt(15.3,15.8,28.7);`
6	Any type Any type		BOOL	**ne**: "not equal" (1 aus 2) `TRUE, if in1 <> in2;` `FALSE, if in1 = in2` `Result:= ne(in1,in2);` `Examples:` `TRUE= ne('Text 1','Text 2');` `FALSE= ne(15.3,15.3);`

Remark:
- "Any type": any elemental datatype BOOL/INT/DINT/UDINT/DWORD/REAL/LREAL/TIME/STRING
- The number of inputs of function blocks "gt", "ge" "eq", "le" and "lt" is free to be altered. To alter the number of inputs double click on the function block and change the "In"-variable under I/O-connectors by entering the desired number oder clicking the arrows up / down.

4.2 Basic FBs (basic function blocks)

Function blocks (FBs) have as many in- and output parameters as needed, which are clearly defined. Furthermore, they can use internal variables, i.e. they have a memory. A counter is a good example for a function block. The counter can be used by one task or by several tasks as well and with a different data set in each case.



Overview "Basic FBs", function blocks
iba's basic function blocks are devided into the following groups:

- Register/Multiplexer
- Edge Detection
- Counter
- Timer/Time functions
- Analytic
- Communication
- Signal Processing
- Debug- and helping function blocks
  - Multi channel Oscilloscope
  - Logical Analyzer
  - Oscilloscope
  - Manual Switch
  - Manual Slider
  - Show String Value
  - Dat File Write
  - Dat File Cleanup

4

4.2.1. Register / Multiplexer

Registers are storage elements. If the control input "set" is TRUE the value of input "value" will be stored and forwarded to the output. Any alternation of the input value will only be taken as long as "set" is TRUE. If the input "reset" is TRUE, the output will be resetted. The control input "set" dominates "reset". (see timing-diagram below)

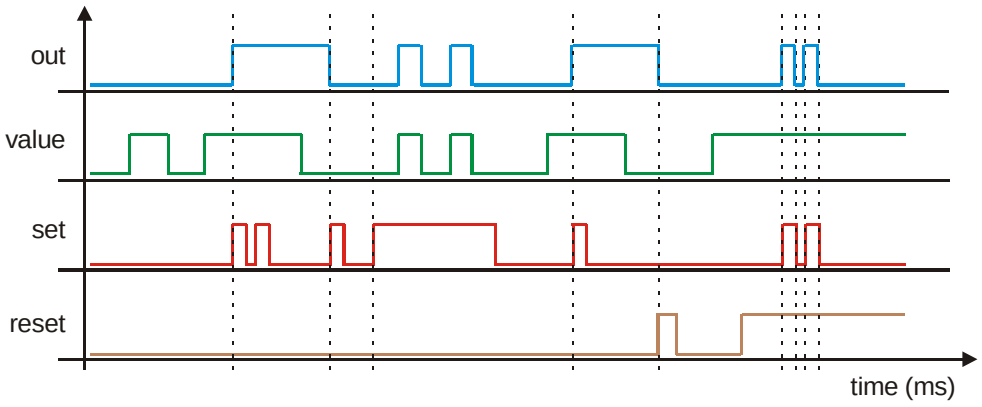
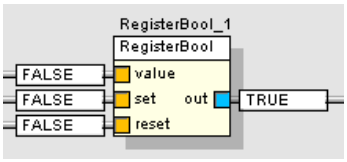
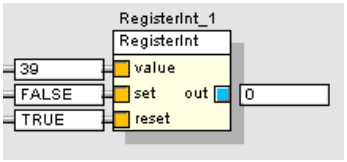
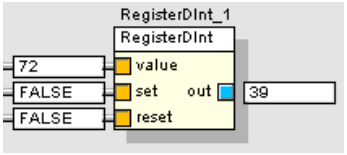
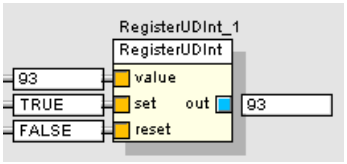
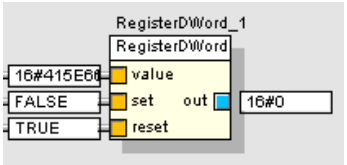
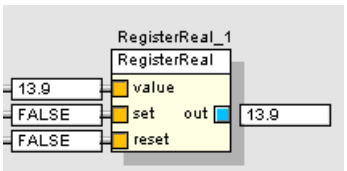
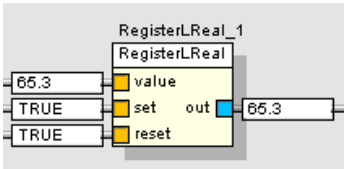
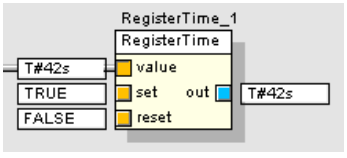
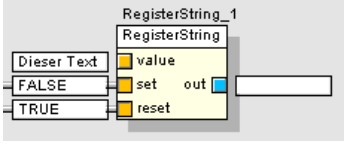
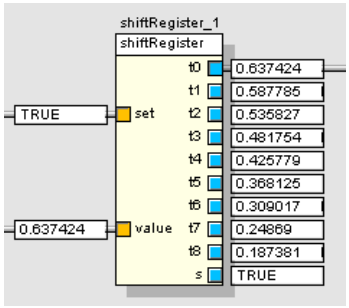
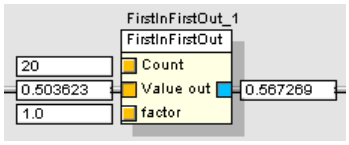
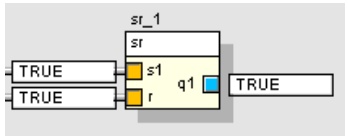
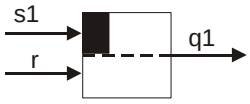
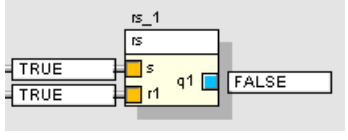
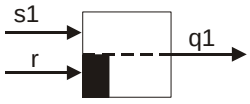


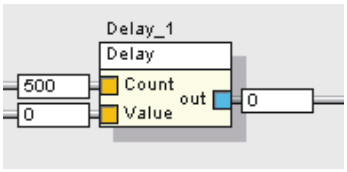
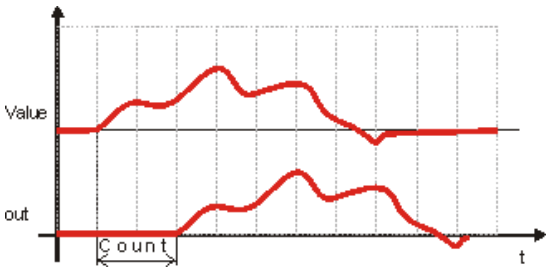
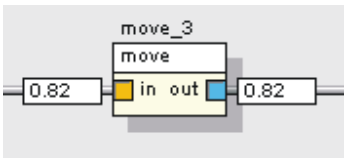
Fig. 69 Timing diagram of register / multiplexer function blocks

4.2.1.1. Register function blocks

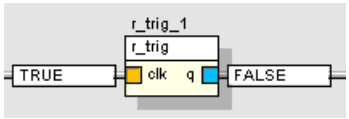
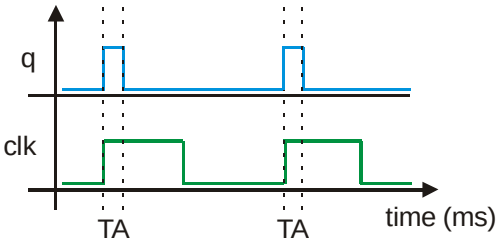
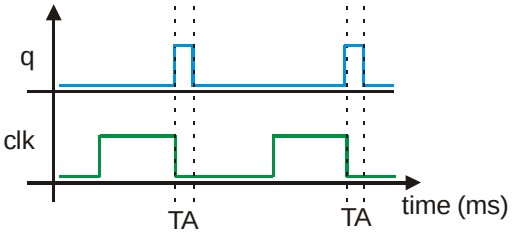
No.	Source Type	Register Function Blocks Symbol	Target Type	Description, Examples
1	BOOL BOOL BOOL		BOOL	**RegisterBool**: Store data type BOOL `Result:= RegisterBool(value, set, reset);` `Examples: see Timing-Diagramm`
2	INT BOOL BOOL		INT	**RegisterInt**: Store data type INT `Result:= RegisterInt(value, set, reset);` `Examples: see Timing-Diagramm`
3	DINT BOOL BOOL		DINT	**RegisterDInt**: Store data type DINT `Result:= RegisterDInt (value, set, reset);` `Examples: see Timing-Diagramm`
4	UDINT BOOL BOOL		UDINT	**RegisterUDInt**: Store data type UDINT `Result:= RegisterUDInt(value, set, reset);` `Examples: see Timing-Diagramm`
5	DWORD BOOL BOOL		DWORD	**RegisterDWord**: Store data type DWORD `Result:= RegisterDWord(value, set, reset);` `Examples: see Timing-Diagramm`
6	REAL BOOL BOOL		REAL	**RegisterReal**: Store data type REAL `Result:= RegisterReal (value, set, reset);` `Examples: see Timing-Diagramm`
7	LREAL BOOL BOOL		LREAL	**RegisterLReal**: Store data type LREAL `Result:= RegisterLReal (value, set, reset);` `Examples: see Timing-Diagramm`
8	TIME BOOL BOOL		TIME	**RegisterTime**: Store data type TIME `Result:= RegisterTime (value, set, reset);` `Examples: see Timing-Diagramm`
9	STRING BOOL BOOL		STRING	**RegisterString**: Store data type STRING `Result:= RegisterString (value, set, reset);` `Examples: see Timing-Diagramm`

4.2.1.2. Shift-register and FIFO function blocks

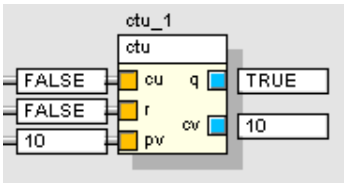
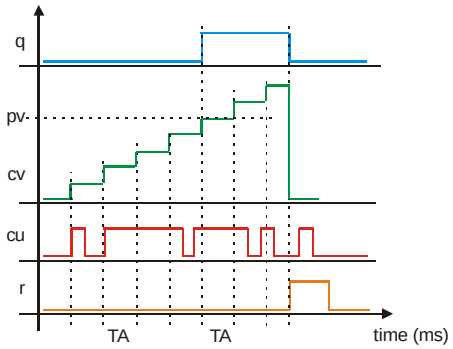
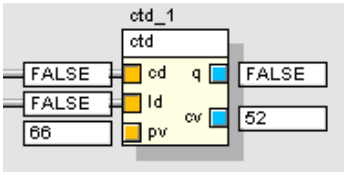
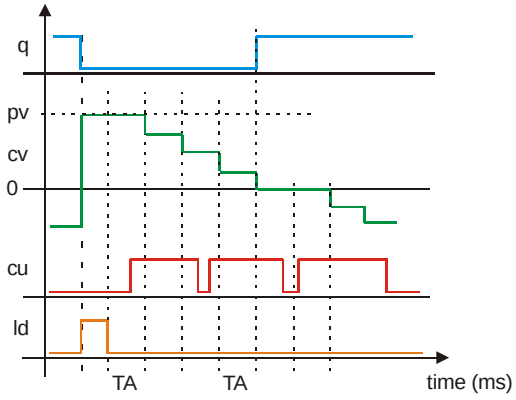
No.	Source Type	Shift-register and FIFO Function Blocks Symbol	Target Type	Description, Examples
1	BOOL LREAL		LREAL LREAL LREAL LREAL LREAL LREAL LREAL LREAL BOOL	**shiftRegister**: Store data type REAL/LREAL in a shift-register t0...t8 `shift, if "set" = TRUE` `t0:= value(t[n])current cycle` `t1:= value(t[n-1]) last cycle` `t2:= value(t[n-2])` `.......` `t8:= value(t[n-8])` `with "n" = task cycle`
2	DINT LREAL LREAL		LREAL	**FirstInFirstOut**: Store data type REAL/LREAL in a FIFO-register `In each task cycle the FIFO-register is shifted by one position. The input "value", multiplied by "factor" is stored in the shift-register.` `Count: length of FIFO-registers` `Value: input value` `factor: multiplier for input "value"`
3	BOOL BOOL		BOOL	**sr**: RS-Flip-Flop (bistable set-dominant)  Truth table Input Values / Output s1 | r | q1 0 | 0 | q1 0 | 1 | FALSE 1 | 0 | TRUE 1 | 1 | TRUE
4	BOOL BOOL		BOOL	**rs**: RS-Flip-Flop (bistable reset-dominant)  Truth table Input Values / Output s1 | r | q1 0 | 0 | q1 0 | 1 | FALSE 1 | 0 | TRUE 1 | 1 | FALSE

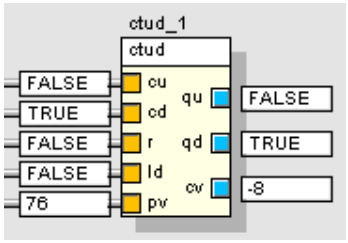
No.	Source Type	Shift-register and FIFO Function Blocks Symbol	Target Type	Description, Examples
5	DINT Any type		Any type	**Delay**: Delay value forwarding The output value "out" follows the input value "value" with a delay of the time equivalent to the number of cycles, given at input "Count".  The function block has a limited memory capacity which applies when using the data type ARRAY ('Value' and 'out'). If the number of array eöements exceeeds 64 then the range of delay values (65536) will be reduced accordingly.
6	Any type		Any type	**move**: Feedback register The output value "out" is an exact copy of the input value "in". This function block is used to define a starting point for evaluation inside a closed loop. The function block grants that the evaluation of the loop always starts at the same place in terms of signal flow for the purpose of a clear evaluation order.

4.2.2. Edge Detection

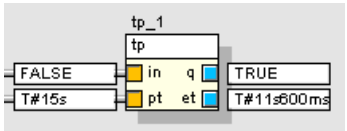
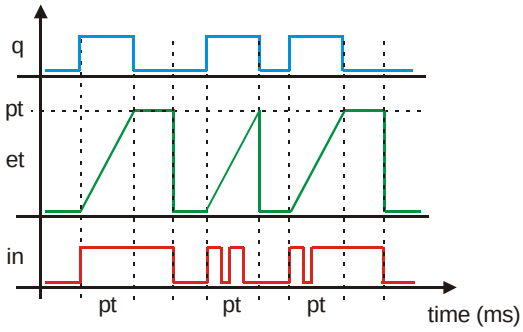
No.	Source Type	Edge Detection Function Blocks Symbol	Target Type	Description, Examples
1	BOOL		BOOL	**r_trig**: Rising Edge Detector If rising edge at input "clk" (0->1)output "q" is set on TRUE for one task cycle. If the input signal clk is TRUE in the moment of switching on the system, the function block generates an impulse (output q = TRUE for one task cycle). 
2	BOOL		BOOL	**f_trig**: Falling Edge Detector If falling edge at input "clk" (1->0)output "q" is set on TRUE for one task cycle. If the input signal clk is FALSE in the moment of switching on the system, the function block generates an impulse (output q = TRUE for one task cycle). 

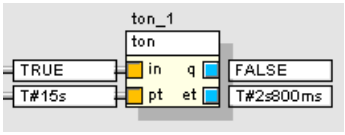
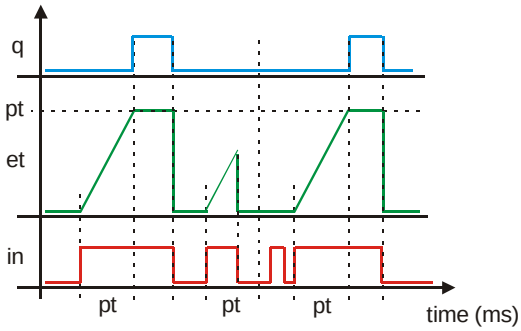
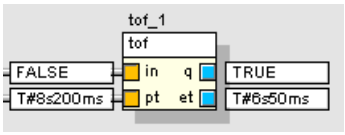
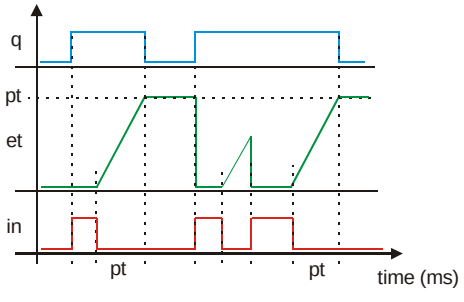
4.2.3. Counter

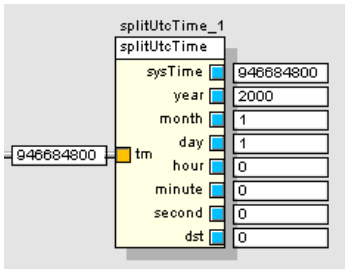
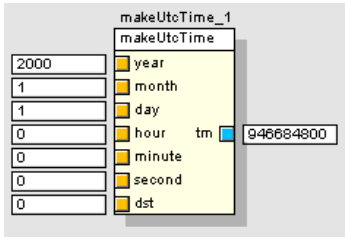
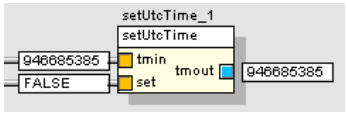
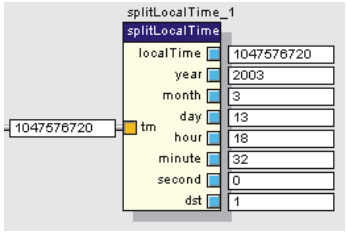
No.	Source Type	Counter Function Blocks Symbol	Target Type	Description, Examples
1	BOOL BOOL DINT		BOOL DINT	**ctu**: Up-Counter If input "cu" is TRUE the counter value "cv" is incremented by one unit per task cycle. When output "cv" has matched the preset value "pv", the output "q" is set TRUE. Input "r" = TRUE resets the counter. 
2	BOOL BOOL DINT		BOOL DINT	**ctd**: Down-Counter If input "Id" is set TRUE the counter value "cv" will be set to preset value "pv". When input "cd" is set TRUE the down-counting starts by decrement of one unit per task cycle. When the counter value "cv" is <= 0 the output "q" is set TRUE. 

No.	Source Type	Counter Function Blocks Symbol	Target Type	Description, Examples
3	BOOL BOOL BOOL BOOL DINT		BOOL BOOL DINT	**ctud**: Up-Down-Counter If input "cu" is TRUE the counter value "cv" is incremented by one unit per task cycle. When output "cv" has matched the preset value "pv", the output "qu" is set TRUE. (see sequence diagram "ctu"-FB) If input "Id" is set TRUE the counter value "cv" will be set to preset value "pv". When input "cd" is set TRUE the down-counting starts by decrement of one unit per task cycle. When the counter value "cv" is <= 0 the output "qd" is set TRUE. (see sequence diagram "ctd"-FB) Input "r" = TRUE resets the counter.

4.2.4. Timer / Time functions (Zeitfunktionen)

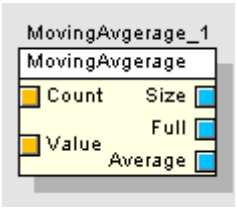
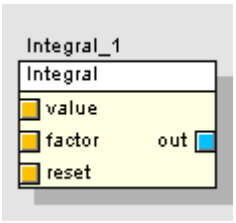
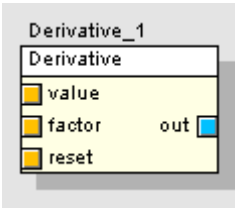
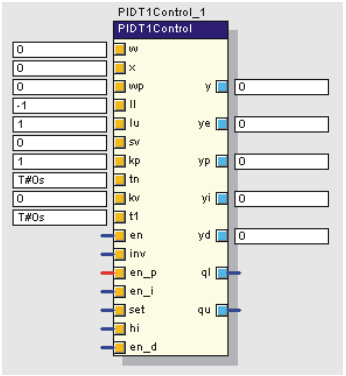
No.	Source Type	Timer Function Blocks Symbol	Target Type	Description, Examples
1	BOOL TIME		BOOL TIME	**tp**: Pulse Timer (pulse extention) The rising edge at input "in" will cause the output "q" to be set on TRUE for the pulse time of "pt". As long as the pulse time is running output "q" cannot be resetted. Output "et" shows the lapsed time. 

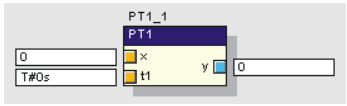
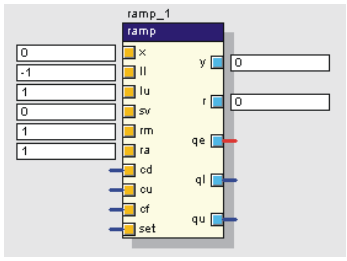
No.	Source Type	Timer Function Blocks Symbol	Target Type	Description, Examples
2	BOOL TIME		BOOL TIME	**ton**: On-Delay The rising edge at input "in" starts the delay time counter for the time "pt". After lapse of "pt" the output "q" is set TRUE until input "in" is FALSE. The output "q" will not be set TRUE if the actual time of "in" being TRUE is shorter than the delay time "pt". Output "et" shows the lapsed time. 
3	BOOL TIME		BOOL TIME	**tof**: Off-Delay If input "in" is TRUE, the output "q" is set TRUE. The falling edge at input "in" starts the delay time counter for the time "pt". After lapse of "pt" the output "q" is set FALSE. The output "q" will remain unchanged if the actual time of "in" being FALSE is shorter than the delay time "pt". Output "et" shows the lapsed time. 

No.	Source Type	Timer Function Blocks Symbol	Target Type	Description, Examples
4	UDINT		UDINT DINT DINT DINT DINT DINT DINT	**splitUtcTime**: segmentation UTC-time This function block converts the input "tm" (given as UTC-time in seconds)to the output variables year, month, day, hour, minute and second. UTC-time is the number of seconds lapsed since 1970-01-01, 00:00:00. Examples: tm= 1; 01.01.1970/00:00:01 tm= 2_592_000 31.01.1970/00:00:00 tm= 946_684_800 01.01.2000/00:00:00 equal to 30 years (60*60*24*365) plus 7 leap days (60*60*24)
5	DINT DINT DINT DINT DINT DINT DINT		UDINT	**makeUtcTime**: generation of UTC-time This function block generates the UTC-time based on the inputs year, month, day, hour, minute and second Examples: 01.01.2000/00:00:00 tm= 946_684_800 08.06.2000/12:00:00 tm= 960_462_000
6	UDINT BOOL		UDINT	**setUtcTime**: Set UTC-time This function block sets the UTC-time.
7	UDINT		UDINT DINT DINT DINT DINT DINT DINT DINT	**splitLocalTime**: Splitting the local system time This function block converts the input value 'tm' (local system time given in seconds)into the output values year, month, day, hour, minute and second plus the information about daylight saving time (dst).

4

4.2.5. Analytic Functions

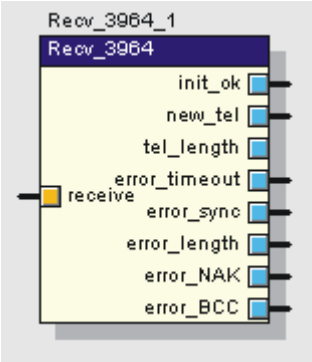
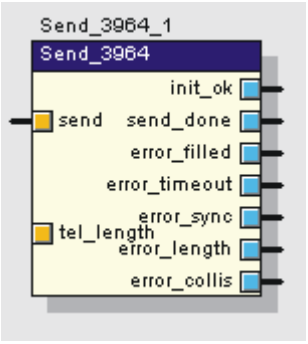
No.	Source Type	Analytic Function Blocks Symbol	Target Type	Description, Examples
1	DINT REAL		DINT BOOL REAL	**MovingAverage**: Cummulated average Input value "Count" sets a number of values to be used as a base for average calculation of input value "Value". Output value "Size" reflects the number of values used for average calculation. Output value "Full" is set TRUE if number of values to cummulate is reached. Output value "Average" returns the cummulated average. Average calculation is done continuously. Input "Count" can be altered during operation.
2	REAL REAL BOOL		REAL	**Integral**: Integrate value over time Output value "out" is the integral of Input "value" multiplied by "factor" over time. "reset" = TRUE resets the output.
3	REAL REAL BOOL		REAL	**Derivative**: Derivate value over time Output value "out" is the derivate of input "value" multiplied by "factor" over time. "reset" = TRUE resets the output.
4	LREAL LREAL LREAL LREAL LREAL LREAL LREAL TIME LREAL TIME BOOL BOOL BOOL BOOL BOOL BOOL BOOL		LREAL LREAL LREAL LREAL LREAL BOOL BOOL	**PIDT1Control**: PIDT1-controller block Universal PIDT1-controller with several modes of operation as P-, I-, PI-, PIDT1-controller. Functions: • Setting start value for integrator • Holding current value of integrator • Precontrol value wp • Control limits ll (low) and lu (up) • Proportional coefficient kp • Reset time tn • Control deviation reversible • Indication of limit violation • Indication of control deviation • Indication of controller output value For more information please refer to chapter **4.2.9**

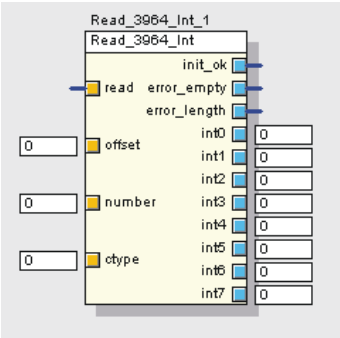
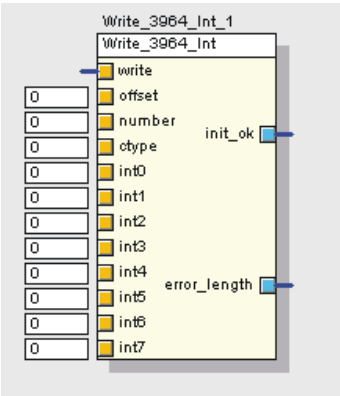
No.	Source Type	Analytic Function Blocks Symbol	Target Type	Description, Examples
5	LREAL TIME		LREAL	**PT1**: Delay function of 1st order The input value 'x' is dynamically delayed by computation with smothing time constant 't1'. The result is copied on the output 'y'. Implementation: `t1_t0 := time_to_lreal(t1) / time_to_lreal(EvalDeltaTime);` `y := 1.0 / (1.0 + t1_t0) * (x + t1_t0 * y_old);` `y_old := y;`
6	LREAL LREAL LREAL LREAL LREAL LREAL BOOL BOOL BOOL BOOL		LREAL LREAL BOOL BOOL BOOL	**Ramp**: Ramp function block The ramp function block provides two different ramps, manual and automatic mode of operation. Functions: • Reference value limitation ('ll' and 'lu') • Going to new reference value via ramp • Setting reference value • Indication of limit violation For more information please refer to chapter **4.2.9**

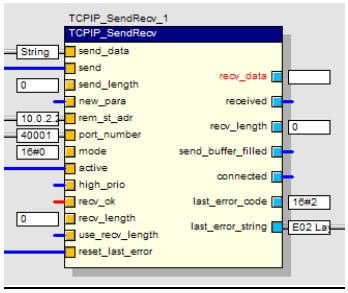
4

4.2.6. Communication Functions

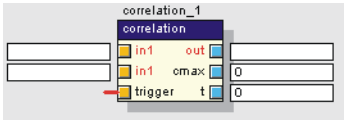
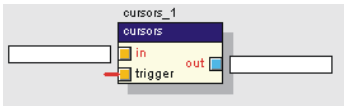
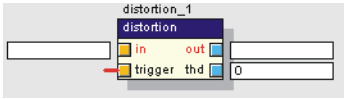
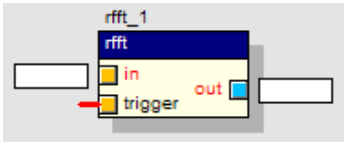
This group of function blocks is dedicated to the communication by serial interface 3964R protocol (DUST) and TCP/IP.

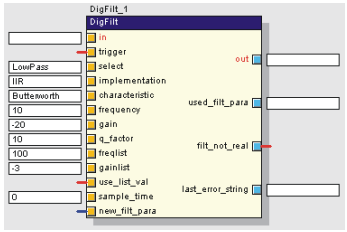
No.	Source Type	Communication Function Blocks Symbol	Target Type	Description, Examples
1	BOOL		BOOL BOOL DINT BOOL BOOL BOOL BOOL BOOL	**Recv_3964**: Receiving a 3964R-telegram (administrativ function block) This function block should always precede a "Read_3964_xxx"-function block. If input "receive" is TRUE the function block tries to receive a telegram from the 3964R-driver. The output "init_ok" is set TRUE if the 3964R-driver has been properly initialized. If a new telegram has been received successfully the output "new_tel" is set TRUE. The output "tel_lentgh" returns the length of the recceived telegram in bytes. The error outputs "error..." are set TRUE if the corresponding error occurred: timeout, synchronization, telegram length (too long), NAK or BCC.
2	BOOL DINT		BOOL BOOL BOOL BOOL BOOL BOOL BOOL	**Send_3964**: Sending a 3964R-telegram (administrativ function block) This function block should always be preceded by a "Write_3964_xxx"-function block. If the input "send" is set TRUE, then the function block tries to submit a telegram of the length given at input "tel_length" to the 3964R-driver. The output "init_ok" is set TRUE if the 3964R-driver has been properly initialized. If the telegram was submitted successfully, then the output "send_done" is set TRUE. The error outputs "error..." are set TRUE if the corresponding error occured: buffer full, timeout, synchronization, telegram length (too short) or collision.

No.	Source Type	Communication Function Blocks Symbol	Target Type	Description, Examples
3	BOOL DINT DINT DINT	Example: Read_3964_Int  	BOOL BOOL BOOL DINT DINT DINT DINT DINT DINT DINT DINT	**Read_3964_Int**: Reading a 3964R-telegram and extraction of a maximum of eight integer values, beginning at offset This function block should always be preceded by a "Recv_3964"-function block. If the input "read" is set TRUE then the function block tries to read integer data from the received telegram. The input "offset" declares the offset for the integer data range in the telegram. "number" defines the number of values to be read (1...8). The input "ctype" is to be used for further specification of the expected data type, e.g. if swapping is required: 0 (default) 4 bytes 2 2 bytes 3 2 bytes and Swap 4 4 bytes 5 4 bytes and Swap (The read- and send function blocks for INT-, UINT and WORD data permit 2- and 4-byte types; the read- and send function blocks for FLOAT only permit 4-byte type) The output "init_ok" is set TRUE if the 3964R-driver has been properly intialized. TRUE at output "error_empty" shows that the receive-buffer is empty and no data are available for reading. Output "error_lenght" is set TRUE if the telegram is too short. The outputs "int0"..."int7" contain the extracted integer values. For the function blocks Read_3964_Uint, Read_3964_Word and Read_3964_Float, the rules apply correspondingly.
4	BOOL DINT DINT DINT DINT DINT DINT DINT DINT DINT DINT DINT	Example: Write_3964_Int  	BOOL BOOL	**Write_3964_Int**: Packaging of up to eight integer values into a 3964R-telegram This function block should always precede a "Send_3964"-function block in order to send the data. If the input "write" is set TRUE then the function block tries to write the "number" of integer values which are given at the inputs "int0"..."int7" into a 3964R-telegram considering the "offset" and the type specification at "ctype". (For information about "ctype" please<refer to description Read_3964_int above.) The output "init_ok" is set TRUE if the 3964R-driver has been properly initialized. The output "error_length" is set TRUE if the telegram is too short to contain the values. For the function blocks Write_3964_Uint, Write_3964_Word and Write_3964_Float, the rules apply correspondingly.

No.	Source Type	Communication Function Blocks Symbol	Target Type	Description, Examples
5	untyped BOOL UDINT BOOL STRING UDINT BOOL BOOL BOOL UDINT BOOL BOOL		untyped BOOL UDINT BOOL BOOL DWORD STRING	**TCPIP_SendRecv**: Sending and receiving data via TCP/IP This function block may be used instead of DLL-based communication functions. Input parameters: *Send_data*: Data to be sent (data types String or Array) *send*: Send command; every task cycle when this input is TRUE, the function tries to send. *Send_length*: Number of bytes to be sent. If = 0, either the entire array or the connected string will be sent. If the value exceeds the total array length the length will be limited to array length. *New_para*: Accepting new function block parameters if = TRUE. *Rem_st_Adr*: Remote Station Address. IP-address of the target PC which is the communication partner. Parameter format: nnn.nnn.nnn.nnn. Value is only required when the instance of the function block is activ on the TCP/IP connection. *Port_number*: Port-number of the connection *Mode*: Bit 0 = 0: Strings will terminate to a value of 0. Bit 0 = 1: Strings will not terminate. Bit 1 = 0: Readbuffer will not be deleted after reading. Bit 1= 1: Readbuffer will be delete after reading. *Active*: This instance of the function block is activ on the TCP/IP-connection if = TRUE. *High_prio*: High Priority Mode = TRUE, for fast TCPIP-communication < 10 ms cycle time. *Recv_ok*: Controls data flow on receiver-side. If = TRUE data reception in the task is possible. *Recv_length*: Length of received messages (only in conjunction with Use_recv_length) *Use_recv_length*: If = TRUE, messages of length Recv_Length will be received. *Reset_last_error*: Reset error outputs Output parameters: *Recv_data*: Received data (data types String or Array) *Received*: Status; If = TRUE, a new message has been received in the current task cycle. *Recv_length*: Number of received bytes *Send_buffer_filled*: If = TRUE, a send trial failed because the first send buffer was still filled on first level. *Connected*: If = TRUE, the connection has been established. *Last_error_code*: Code of the error which occurred recently. *Last_error_string*: Text of the error which occurred recently.

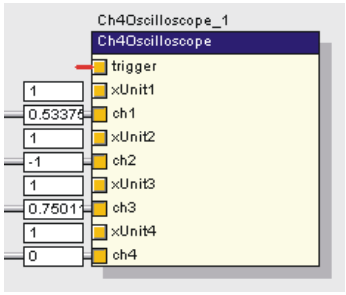
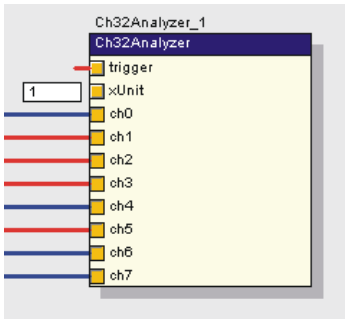
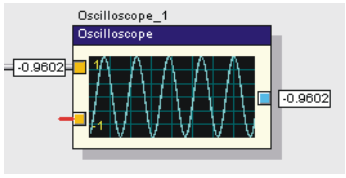
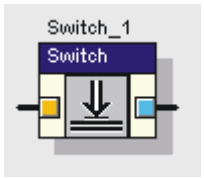
4.2.7. Signal processing

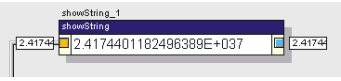
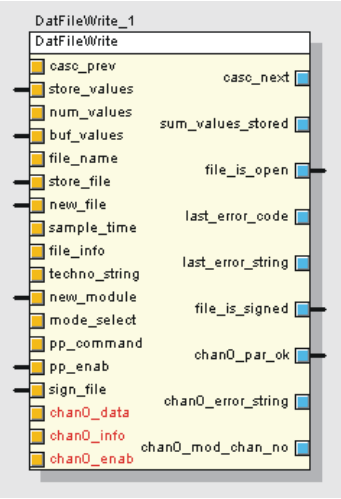
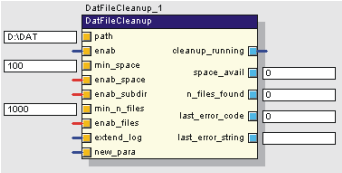
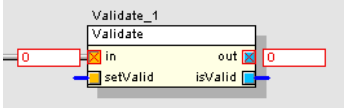
No.	Source Type	Signal Processing Function Blocks Symbol	Target Type	Description, Examples
1	ARRAY ARRAY BOOL		ARRAY REAL DINT	**Correlation**: Correlation of one or two signals This function block evaluates the cross-correlation between two Signals or – if the signal level of one of the input signals is too low – the auto-correlation of one signal. Additional outputs are the maximum correlation coefficient and the array index. *in1, in2, out*: One-dimensional arrays with 2, 4, 8, 16, 32, ...65536 elements, Startindex 0 The function block will only be evaluated if *trigger* is TRUE.
2	ARRAY BOOL		ARRAY	**Cursors**: Basic frequency and harmonics Using the method of amplitude comparison this function block evaluates the basic frequency and the corresponding harmonis of input signal *in*. The basic frequency can be found at index 0 in the output array *out*, the harmonic at the indices 1...n. *in, out*: One-dimensional REAL-Arrays with 2, 4, 8, 16, 32,...65536 elements, Startindex 0 The function block will only be evaluated if *trigger* is TRUE.
3	ARRAY BOOL		ARRAY REAL	**Distortion**: Grade of distortion This function block evaluates the grade of distortion (harmonic distortion) of an input signal *in* and the total harmonic distortion (thd). *in, out*: One-dimensional REAL-Arrays with 2, 4, 8, 16, 32,...65536 elements, Startindex 0 The function block will only be evaluated if *trigger* is TRUE.
4	ARRAY BOOL		ARRAY	**rfft**: Real Fast Fourier Transformation This function block returns a single-sided fft result (absolute value). Input *in* should be of data type array, e.g. an array of reals with dimension of 2^n (64...32768 array indexes). Output *out* is also an array of reals but with dimension of $2^{(n-1)}$ (e.g. 32...16384). Input *trigger* = TRUE enables the FFT-calculation. If *trigger* is FALSE the function block doesn't calculate and so won't consume processor time.

No.	Source Type	Signal Processing Function Blocks Symbol	Target Type	Description, Examples
5	UNTYPED BOOL STRING STRING STRING LREAL LREAL LREAL ARRAY ARRAY BOOL LREAL BOOL		UNTYPED STRING BOOL STRING	**DigFilt**: digital filter Digital filter for continuous or buffered signals connected to input *in*. Lowpass, highpass, bandpass and bandstop filtertypes are available. Filter implementation may either be IIR- (Infinite Impulse Response) or FIR- (Finite Impulse Response). As IIR-Filter Butterworth-, Tschebyscheff-, Elliptic- and Invers Tschebyscheff characteristics are available. As FIR-Filter the windowing types Rectangle, Bartlett, Blackman, Hamming, Hanning and Kaiser are available. Setup of parameters and errors will be indicated as text (*used_filt_para, last_error_string*). For more information please refer to chapter 4.2.9

4

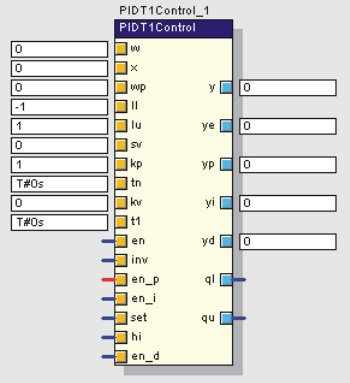
4.2.8. Special and helpful basic FBs

No.	Source Type	Special Basic Function Blocks Symbol	Target Type	Description, Examples
1	BOOL REAL Any type REAL Any type REAL Any type REAL Any type			**Ch4Oscilloscope**: Multichannel Oscilloscope Function block to use like a probe. Scalable from one up to four channels. TRUE at input *trigger* starts the monitoring. Inputs *xUnitn* for x-scale, inputs *chn* for signals to be monitored. Open display with tool bar button or right mouseclick on function block. For more information please refer to chapter 3.13.4
2	BOOL REAL BOOL BOOL BOOL BOOL BOOL BOOL BOOL			**Ch32 Logic Analyzer**: Oscilloscope for boolean signals Function block to use like a probe. Scalable from one up to 32 channels. TRUE at input *trigger* starts the monitoring. Input *xUnit* for x-scale, inputs *chn* for boolean signals to be monitored. Open display with tool bar button or right mouseclick on function block (multichannel oscilloscope) For more information please refer to chapter 3.13.4
3	Any type BOOL		REAL	**Oscilloscope**: ordinary oscilloscope This function block monitors shape and trend of one signal. Lower input (BOOL) on TRUE will enable continuous autoscale function. Output (REAL) shows the last value. Attention: Processing time consumption depends on size of representation in function block diagram! The bigger the display the more time it needs!
4	BOOL		BOOL	**switch**: pushbutton and switch This function block is a good help for manual simulation of boolean signals. For pushbutton-function use left mouseclick on function block (symbol). Output will be TRUE as long as mouse button is pressed. For switch-function use right mouseclick. Output toggles with every click. Input for alternativ switch control by a boolean signal. If input is TRUE (permanent or impulse) then output of switch is TRUE (permanent, switching off manually).

No.	Source Type	Special Basic Function Blocks Symbol	Target Type	Description, Examples
5	REAL REAL		REAL DINT	**slider**: digital potentiometer This function block returns any value in the range given by minimum value (upper input) and maximum value (lower input)depending of the slider position. Resolution is 1000 steps. The slider position is returned at the lower output (0...1000). Inputs are set on 0 / 1 by default but they can have any value.
6	Any type		STRING	**showString**: display of any value This function block is helpful for display of any value, particularly for long figures or strings. Interprets input always as string. Output is of type STRING.
7	DWORD BOOL DINT BOOL STRING BOOL BOOL LREAL STRING STRING BOOL DWORD STRING BOOL BOOL Any type Any type Any type		DWORD DINT BOOL DWORD STRING BOOL BOOL STRING STRING	**DatFileWrite**: Creating and filling of *.dat-files This function block is designed to open, to fill and to close data files of iba's *.dat-type directly in the ibaLogic layout. As usual the created data files can be further processed and evaluated with ibaAnalyzer or other tools, which are able to read the dat-format. Due to the function block's complex functionality please refer to the following chapter 4.2.9 for detailed information.
8	STRING BOOL UDINT BOOL BOOL UDINT BOOL BOOL BOOL		BOOL UDINT UDINT DINT STRING	**DatFileCleanup**: Clean up the harddisk This function enables the ibaLogic application to care out a cleanup-strategy in terms of old data files on the harddisk. Depending on settings and criteria (input parameters) similar to those in ibaPDA old data files may be deleted or overwritten. For more information please refer to chapter 4.2.9
9	Any type BOOL		Any type BOOL	**Validate**: Monitoring and setting valid signals This function block monitors the validity of a connected input signal. Output *isValid* is TRUE if input *in* is valid. If input *in* is invalid then the output *out* is invalid too and the output *isValid* is FALSE. If input *setValid* is set TRUE then output *out* is forced to valid, with the recent value. By using this function block in a network of recursive evaluations (loops) it's possible to prevent an invalid deadlock of the evaluation. Just insert this block in the loop and set the input *setValid* =TRUE.

4.2.9. Complex funktion blocks

4.2.9.1. PIDT1Control



Function and usage

Universal PIDT1-controller with several modes of operation as P-, I-, PI-, PIDT1-controller.

Functions:

- ❑ Setting start value for integrator
- ❑ Holding current value of integrator
- ❑ Precontrol value wp
- ❑ Control limits ll (low) and lu (up)
- ❑ Proportional coefficient kp
- ❑ Reset time tn
- ❑ Control deviation reversible
- ❑ Indication of limit violation
- ❑ Indication of control deviation
- ❑ Indication of controller output value (P, I, DT1)

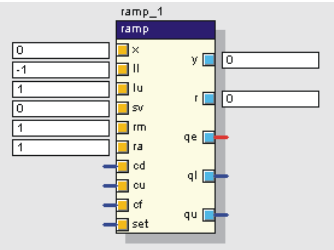
Connectors

Connector	Data type	Description
w	LREAL	Reference value
x	LREAL	Actual value
wp	LREAL	Precontrol value
ll	LREAL	lower limit
lu	LREAL	upper limit
sv	LREAL	Initial value
kp	LREAL	P-gain
tn	TIME	Reset time
kv	LREAL	D-gain

cont'd. PIDT1

Connector	Data type	Description
tl	TIME	D-time constant
en	BOOL	Controller release
inv	BOOL	Inversion of control deviation
en_p	BOOL	Enable P-controller mode
en_i	BOOL	Enable I-controller mode
set	BOOL	Set integrator
hi	BOOL	Hold integrator
en_d	BOOL	Enable D-controller
y	LREAL	Control value
ye	LREAL	Control deviation
yp	LREAL	Output value P-controller
yi	LREAL	Output value I-controller
yd	LREAL	Output value D-controller
ql	BOOL	lower limit reached
qu	BOOL	upper limit reached

4.2.9.2. Ramp



Function and usage

The ramp function block provides two different ramps, manual and automatic mode of operation.

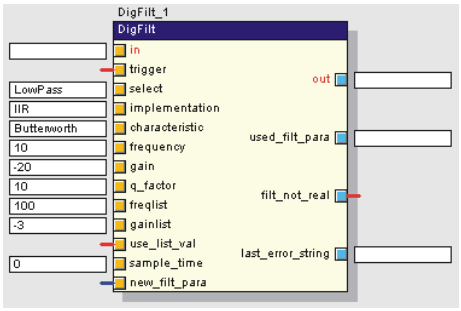
Functions:

- ❑ Reference value limitation ('ll' and 'lu')
- ❑ Going to new reference value via ramp
- ❑ Setting reference value
- ❑ Indication of limit violation

Connectors

Connector	Data type	Description
x	LREAL	Input value (reference value)
ll	LREAL	Lower limit
lu	LREAL	Upper limit
sv	LREAL	Initial value
rm	LREAL	Manual ramp (10/s)
ra	LREAL	Automatic ramp (10/s)
cd	BOOL	Descending ramp (manual ramp control)
cu	BOOL	Ascending ramp (manual ramp control)
cf	BOOL	Ramp acc. to. input value (automatic ramp control)
set	BOOL	Set output value
y	LREAL	Output value; $y_n = y_{n-1} + T_a \bullet r \bullet 10$ T_a = task cycle time r = used ramp
r	LREAL	Used ramp (1/s)
qe	BOOL	Output value = input value
ql	BOOL	lower limit reached
qu	BOOL	upper limit reached

4.2.9.3. DigFilt - digital filtering of signals



Function and usage

This function blocks works like a digital filter for continuous or buffered signals.

Signals to be measured may be cleared of disturbing frequencies (noise or hum) in order to improve the control quality of a connected open or closed-loop control. In conjunction with the rfft function block the frequencies which are included in a signal may be detected and filtered out.

Connectors

Connector	Data type	Description
in	untyped	Input signal to be filtered; permissible data types: REAL and one-dimensional ARRAY of REAL
trigger	BOOL	The function block will only be evaluated if *trigger* is TRUE.
select	STRING	Selction of filter type; the input string must have the exact spelling as follows (high- and low case sensitive): LowPass.....for lowpass filter HighPass.....for highpass filter BandPass.....for bandpass filter BandStop.....for bandstop filter (Error message no. E00 in case of misspelling)
implementation	STRING	Selection of filter implementation; the input string must have the exact spelling as follows (high- and low case sensitive): IIR.....(Infinite Impulse Response) FIR....(Finite Impulse Response) This input depends on the selection at input *characteristic*. (see table below) (Error message no. E01 in case of misspelling)
characteristic	STRING	Selection of the filter characteristic; the input string must have the exact spelling as follows (high- and low case sensitive): Butterworth, Chebyshev, Elliptic or InvChebyshev (IIR) Rectangular, Bartlett, Blackman, Hamming, Hanning or Kaiser (FIR) This input depends on the selection at input *implementation*. (see table below) (Error message no. E01 in case of misspelling)

cont'd. DigFilt

Connector	Data type	Description
frequency	LREAL	Corner- or main frequency of the filter, given in Hz
gain	LREAL	Attenuation (per decade or maximum), given in dB
q_factor	LREAL	Quality factor, ratio of main frequency and bandwidth (for bandpass- and bandstop filters)
freqlist	ARRAY[0..3] of LREAL	List of filter frequencies An array of up to four frequency values may be connected to this input. The input signal will be filtered on all of these frequencies. Each frequency may be filtered with an individual attenuation. Thus, several frequencies may be filtered from the input signal at the same time.
gainlist	ARRAY[0..3] of LREAL	List of attenuation values, corresponding to the list of filter frequencies.
use_list_val	BOOL	Enable (=TRUE) usage of frequency and attenuation values from the arrays *freqlist* und *gainlist*.
sample_time	LREAL	Sample time in ms which corresponds with the samples of the input signal.
new_filt_para	BOOL	This input must be set TRUE for one task cycle if new filter parameters should apply.
out	untyped	Filter output signal; the data type derives automatically from the input signal.
used_filt_para	STRING	Output / indication of the used filter parameters
filt_not_real	BOOL	If the function was not able to evaluate a filter, e.g. due to an incompliance of input signal and filter parameters, this output is set TRUE.
last_error_string	STRING	Recent error message (text)

Combinations of parameters and their dependence

if "implementation" = ...	....then "characteristic" = ...
IIR	Butterworth, Chebyshev, Elliptic or InvChebyshev
FIR	Rectangular, Bartlett, Blackman, Hamming, Hanning or Kaiser

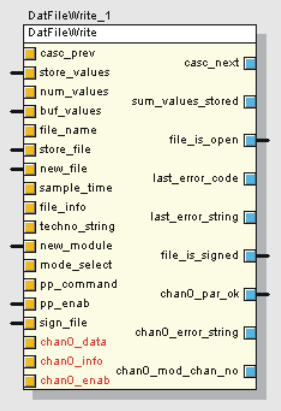
Sample application (Layout) on CD

sample_layout_digfilt_101.lyt

This sample application helps to get familiar with the function and usage of the DigFilt function block. Some support for entering filter parameters (type, implementation, characteristic) is provided.

The sample shows the filtering of a buffered signal (task 0) and a time-discrete signal (task 2) as well.

4.2.9.4. DatFileWrite-function block – generation of iba data files (*.dat)



4

Function and usage

The DatFileWrite function block stores data in dat-files which may be analysed later with ibaAnalyzer or any other offline analysis tool which is able to read the iba dat-file format. The data types of the data that can be stored are INTEGER, REAL or BOOL data or ARRAYs of these types. The data stored per channel in a dat file can be single data or buffered data. Each individual data channel can be enabled and application-specific information can be written to the dat file.

The number of data inputs to the DatFileWrite function block is extensible from minimum 1 to a maximum of 16 input and output groups. For each group a data input, an info input and an enable input together with a para_ok output, last_error_text output and Mod_chan_no output are added.

Connectors

Connector	Data type	Description
casc_prev	DWORD	Not used, reserved for future use
store_values	BOOL	Enable storage; if the file is open data will be stored in the dat file in every evaluation cycle this input is set on TRUE.
num_values	DINT	Number of values to be stored; only used if buffered values are used, the minimum number of stored data per channel, per storing cycle is 1. This value is taken into account every cycle when data is stored.
buf_values	BOOL	Enable use of buffered values; if set on TRUE buffered values are used (this input is taken into account once when a new file is created)
file_name	STRING	Data filename; file name of stored file including drive and path. This value is taken into account once when a new file is created.
store_file	BOOL	Start function block; a rising edge on this input runs the input connector check, opens a file and enables internally the storing of data. A negative edge on that input closes the file and runs the postprocessing command if this function is enabled.

cont'd. DatFileWrite

Connector	Data type	Description
new_file	BOOL	Make a new file; a rising edge on this input closes the currently used and open file and opens a new file using the *file_name* input. In any case the *store_file* input must be set on TRUE. (corresponds to the continuous recording in ibaPDA)
sample_ time	LREAL	Sample time; this input value is used for setting the clk-entry in the dat file and means the time between two samples of a channel in seconds.
file_info	STRING	optional; at the time of closing the file the file info string is used to add user defined entries in the dat file.
techno_string	STRING	optional; at the time of closing the file the technos-tring is inserted in the dat file.
new_module	BOOL	Align to new module; if set on TRUE a new channel will be inserted at the beginning of a new module in the file.
mode_select	DWORD	Control word for miscellaneous functions; the functions described below will be executed if the corresponding bits in the DWORD are TRUE. Bit0: Flush Buffers The contents of the internal data buffer for the online-compression will be written into the dat-file. Thus it's possible to access and analyse these data with ibaAnalyzer even when the file is still open. Bit1: Asynch Access (asynchron access) All file and system calls will be executed on a separate thread (asynchron to the thread of evaluation). For this mode the following restrictions apply: 1. Only one dat-file can be opened by a function block at a time. The current data file must be fully stored before the next file can be opened. 2. The data buffer between task-evaluation (which delivers the data) and the asychron thread (which fills the data into the file) is limited to 1 MB. Bit2...32: Not used, reserved for future use;
pp_command	STRING	Postprocessing command; is executed when a file is closed and at least one sample is stored and the function is enabled.
pp_enab	BOOL	Postprocessing enable; if set on TRUE, then the postprocessing command is enabled.
sign_file	BOOL	If set on TRUE the file will be signed to enable enhanced ibaAnalyzer functions for offline analysis.
chanx_data	untyped	Data input for each channel (x = 0 ... 15)
chanx_info	untyped	Additional info for each channel (x = 0 ... 15)
chanx_enab	untyped	Enable data acquisition for each channel (x = 0...15)

cont'd. DatFileWrite

Connector	Data type	Description
casc_next	DWORD	Not used, reserved for future use
sum_values_stored	DINT	Sum of values stored in the current dat-file per channel. Every time a new dat-file is created, the value is set on 0.
file_is_open	BOOL	Status bit: File is open (= TRUE). Data can only be stored if the file is open.
last_error_code	DWORD	Used for indication what error happened recently (code)
last_error_string	STRING	Used for indication what error happened recently (text)
file_is_signed	BOOL	This flag is set on TRUE when the file is closed and could be signed. It is reset (FALSE) when a new file is opened.
chanx_par_ok	BOOL	Status: Parameter ok for each channel (x = 0 ... 15); When a file is opened, the parameters of the input connectors (*_data*, *_info* and *_enab*) are checked for data types and number of entries. If the check found no error and if the channels are enabled for storing, the *chanx_par_ok* output is set on TRUE.
chanx_error_string	STRING	For each channel (x = 0 ... 15) If the check of the input connectors found an error, a reason is displayed here (text message).
chanx_mod_chan_no	STRING	For each channel (x = 0 ... 15) Indication of module and channel numbers of the signal in the dat-file.

4

How to use the function block

After placing the "DatFileWrite"-function block in a layout the user needs to fill out or specify the sampling time and make a decision whether single values or buffered values should be used. Don't forget to fill out the "num_values" input if you use buffered values mode. Then the signals to be stored must be connected to the *chanx_data* inputs. For each input channel that needs to be stored the *_enable* input must be set on TRUE either by one single boolean input or an matching array. The next step is to specify a file name.

In order to store data, first the file must be opened, and a check of the input channels will be performed. To do that set the *store_file* value on TRUE. If the check for any input channel fails, the related *par_ok* output will is set on FALSE and an error string is generated. You may want to use the "ShowString"-function block to take a look at the reason. Finally you should be able to fix the problem so that the *file_open* output will turn on TRUE.

With the *store_file* input permanently set on TRUE and the *store_values* input set on TRUE, the function block will store data.

When all data are stored in a file set the *store_file* input on FALSE. Then the file will be closed, signed if selected and the postprocessing command may be executed if selected.

cont'd. DatFileWrite

Rules for overloadable input connectors

- ***Chanx_data***
  - Scalar data type INT, REAL or BOOL, if single values are used.
  - One-dimensional array of data type INT, REAL or BOOL, if single values are used every index of the first dimension means one signal, if buffered values are used every index of the first dimension means a different sample of the same signal.
  - Twodimensional array of data type INT, REAL or BOOL, only if buffered values are used. Every index of the first dimension means one signal, every index of the second dimension means a different sample of the same channel.
- ***Chanx_info – optional***
  - STRING, this string is used for every signal (= channel) to add the info entries into the dat files.
  - Array of the same dimension as the data array of any data type (strings can be hidden there, since there are no arrays of strings possible) the number of entries in the first dimension must match the number of entries in the data array.
- ***Chanx_enab –***
  - BOOL, this flag is used for every signal of a channel to enable the storing.
  - One-dimensional array of data type BOOL, can be used with single values or buffered values, the number of signals that can be enabled with this array must match the number of signals in the data input.

Special Remarks

- The cascade inputs and outputs are not used yet.
- The time consuming function calls for storing data in a file are part of the layout evaluation and may block the evaluation of your layout. In order to prevent such problems enable the asynchron access mode (input *mode_select*, bit1 = TRUE).
- The sorting of channels in the ibaAnalyzer supports 32 analog plus 32 digital channels per module. If more than 32 signals should be stored and / or a mix of analog and digital signals is used it is strongly suggested to use the ibaAnalyzer-compliant 32-analog-plus-32-digital-signal arrangement in that order per module.
- In order to use the function block the ibaLogic layout must run in online mode and some iba hardware must be installed so that the ibaLogic driver is working. The function block also works in demo mode with or without dongle. If the function block is used without a dongle the created dat-files won't be signed, i.e. the data may be viewed with ibaAnalyzer but not analyzed. If ibaLogic is used without dongle but in eCon-mode, the dat-files will be created without signature, i.e. the data may be viewed with ibaAnalyzer but not analyzed. If ibaLogic is used with a dongle the function block creates signed dat-files for full analysis capability.

cont'd. DatFileWrite

Rules for text entries in dat-file

Any text entry in the dat file follows the rule <entry_name>:<any_text>. Entries can be made for the file or for an individual signal. The entries are used and displayed in the ibaAnalyzer. The DatFileWrite function block allows to enter multiple entries separated by ',' (comma). Some entry names have a predefined meaning in the dat-file and writing some vital entries in the dat-file will be prohibited by the function block. Some entries will be written by the function block itself only if the user has made no selection.

Liste of global header text entries (excerpt)

Entry_name	Meaning	Class	by ibaLogic	by user
beginheader	Beginning of the header	vital	yes	no
starttime	Starttime of the file	vital	yes	no
clk	Sample distance	vital	yes	no
frames	Number of values	vital	yes	no
typ	Type of file	vital	yes	no
ibalogic	ID of Generator	optional	yes	no
technostring	Technostring information	optional	yes	no
endheader	End of the header	vital	yes	no
module_name_x	Name of the module	optional	no	optional

Liste of channel header text entries (excerpt)

Entry_name	Meaning	Class	by ibaLogic	by user
beginchannel	Beginning of the header	vital	yes	no
channel_offset	Offset of Channel	vital	yes	no
digchannel	Digital Channel info	vital	yes	no
name	Name of Channel	vital	yes	optional
minscale	Minimum Scale	vital	yes	optional
maxscale	Maximum Scale	vital	yes	optional
endchannel	End of the header	vital	yes	no

For storing additionally application-specific information in the dat-file the following method can be used:

Add a string like „myentry:mytext" in the input connector string. More than one entry must be separated by ´,´ (comma).

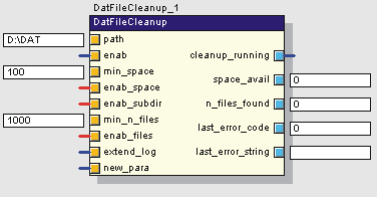
Sample application (layout) on CD

sample_layout_DatFileWrite_301.lyt

This sample application helps to get familiar with the function and usage of the DatFileWrite function block. Some support for parameterize the block is provided.

The sample shows the creation of a dat-file with single signals (Task Sample_1) and buffered signals (Task Sample_2) as well.

4.2.9.5. DatFileCleanup-function block – clean up the harddisk



Function and usage

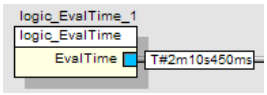
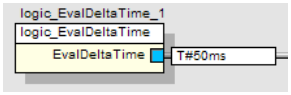
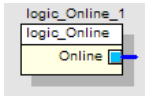
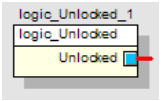
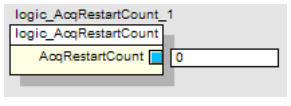
This function block enables the ibaLogic application to care out a cleanup-strategy in terms of old data files on the harddisk. Depending on settings and criteria (input parameters) similar to those in ibaPDA (trigger settings / options) old data files may be deleted or overwritten finally.

Connectors

Connector	Data type	Description
path	STRING	Storage location of the dat-files (= location concerned by cleanup measures); drive name and full path required.
enab	BOOL	The function block will be evaluated with each positive edge at this input; if *enab* is constantly TRUE, the function block is evaluated every 15 min.
min_space	UDINT	Disk space (given in MB) that at least should always be free. If the function block detects a violation of this limit it will start the cleanup measures, provided the *enab_space* input is TRUE.
enab_space	BOOL	Enable keep-minimum-space-function; if set on TRUE the monitoring of free disk space is enabled, see above.
enab_subdir	BOOL	Enable cleanup of empty subdirectories; if this input is set on TRUE empty subdirectories will be removed too after 48 hrs.
min_n_files	UDINT	Minimum number of files to keep; this number determines how many files should stay on the disk. This parameter prevents the system from removing all the dat-files. This situation may occur if other processes, e.g. a PDA-system, writes data to the same harddisk, consuming its free space and violating the lower limit of free disk space.
enab_files	BOOL	Enable (= TRUE) the monitoring of number of files, see above
extend_log	BOOL	Enable (=TRUE) creation of log file to record events during cleanup.
new_para	BOOL	This input must be set on TRUE for one task cycle if new parameters should apply.
cleanup_running	BOOL	Status flag: cleanup is running.
space_avail	UDINT	Free space (MB) during last cleanup
n_files_found	UDINT	Number of files found during last cleanup
last_error_code	DINT	Recent error message (code)
last_error_string	STRING	Recent error message (text)

4.3 Global variables

Generally, ibaLogic is conceptually based on the use of encapsulated data structures. On the contrary to other control applications, global variables are the exception. There are a few global system variables which could be used in function block diagrams, structured text or C++ statements (DLLs).

No.	Variable Name Layout Symbol	Target Type	Description
1	**logic_EvalTime** 	TIME	= time lapsed since start of the application;
2	**logic_EvalDeltaTime** 	TIME	= time lapsed since last start of the task (scan time); the use of this variable will help to eliminate deviations in scan time and to evaluate the correct results.
3	**logic_Online** 	BOOL	= state of layout: online; certain functions or the use of resources may be locked with this variable in dependence of online or Hot-Swap mode of the layout. TRUE: Layout is online, outputs are activ FALSE: Layout is offline, outputs are locked, inputs are still active.
4	**logic_Unlocked** 	BOOL	= state of Layout: unlocked; to be used for locking default values if layout is locked. TRUE: Layout is unlocked, modifications are possible FALSE: Layout is locked, modifications are impossible This variable can be used, for instance, in conjunction with DLLs in order to prevent modification of default values by the DLLs, if not allowed.
5	**logic_AcqRestartCount** 	UDINT	= counter value to indicate the number of driver restarts since start of evaluation. This variable can be used to inform the layout about restarts of drivers (hardware) in order to adjust the hardware parameters if needed.

4.4 Global FBs and macros

Global FBs and macros are to be used when multiple ibaLogic systems should use these functions which are needed in different applications.

If such kind of function or macro blocks had been created by the user as local FBs or macros first, they should then be copied or moved in the Windows Explorer from the folder *...configuration\FBs_Macros* to the folder *...configuration\globalRessource\FBs_Macros*.

- *The same blocks should NOT be available in the local folder and in the global folder at the same time, because they will always be displayed as global FBs and macros.*
- *After deleting or copying of blocks in the folder* ...configuration\globalRessource\FBs_Macros *ibaLogic must be restarted in order to refresh the display of the function tree.*
- *Deleting of FBs/MBs is only permitted in the Windows Explorer (not inside of ibaLogic)!*
- *If the contents of a block has been modified afterwards, this block has to be exported again as a local FB/MB, followed by copying it to the global folder with the Windows Explorer.*

4.5 Global DLLs

Global DLLs which had been created by the user in C or C++ are useful if the functionality of a DLL is needed in multiple projects.

The global DLL is made available in ibaLogic by copy it to the folder *...configuration\globalRessource\DLLs,* using the Windows Explorer.

- *The same DLLs should NOT be available in the local folder and in the global folder at the same time, because they will always be displayed as global DLLs.*
- *After deleting or copying of DLLs in the folder* ...configuration\globalRessource\DLLs *ibaLogic must be restarted in order to refresh the display of the function tree.*
- *Deleting of DLLs is only permitted in the Windows Explorer (not inside of ibaLogic)!*

4.6 Local FBs and Macros

Local FBs and macros are to be used when the functionality of a FB or macro block (MB) is needed multiple times in the same project.

After the project-specific block has been created in the layout it must be exported. In order to export a FB or MB make a right mouseclick on the block in the layout. From the context menü choose ↪*Modify* ↪*Function Block*, resp. *Macro Block* and then click on the *Export* button in the FB-/MB-dialog. The new FB or MB is then available as a file (*.fbm) in the folder *...configuration\FBs_Macros*.

If there are more FBs or MBs already available as files in other projects they can be copied easily with the Windows Explorer to the local folder *...configuration\FBs_Macros*.

- *The same blocks should NOT be available in the local folder and in the global folder at the same time, because they will always be displayed as global FBs and macros.*
- *After deleting or copying of blocks in the folder* ...configuration\globalRessource\FBs_Macros *ibaLogic must be restarted in order to refresh the display of the function tree.*
- *Deleting of FBs/MBs is only permitted in the Windows Explorer (not inside of ibaLogic)!*
- *If the contents of a block has been modified afterwards, this block has to be exported again as a local FB/MB, followed by copying it to the global folder with the Windows Explorer.*

4.7 Local DLLs

Local DLLs are to be used when the functionality of a DLL is needed multiple times in the same project.

In order to use a DLL which had been created by the user in C or C++, it must be made availble in ibaLogic in one of the following ways:

- When ibaLogic is running, use the menu ↪*File* ↪*Open DLL*... A file browser helps finding the DLL-file. Click on the *Open* button and the DLL will be loaded and copied to the folder *...configuration\DLLs*.
- The DLL file may also be copied with Windows Explorer to the folder *...configuration\DLLs* but the DLL is not available in ibaLogic until ibaLogic has been restartet.

- *The same DLLs should NOT be available in the local folder and in the global folder at the same time, because they will always be displayed as global DLLs.*
- *After deleting or copying of DLLs in the folder* ...configuration\DLLs *ibaLogic must be restarted in order to refresh the display of the function tree!*

Deleting of DLLs is only permitted in the Windows Explorer (not inside of ibaLogic)!

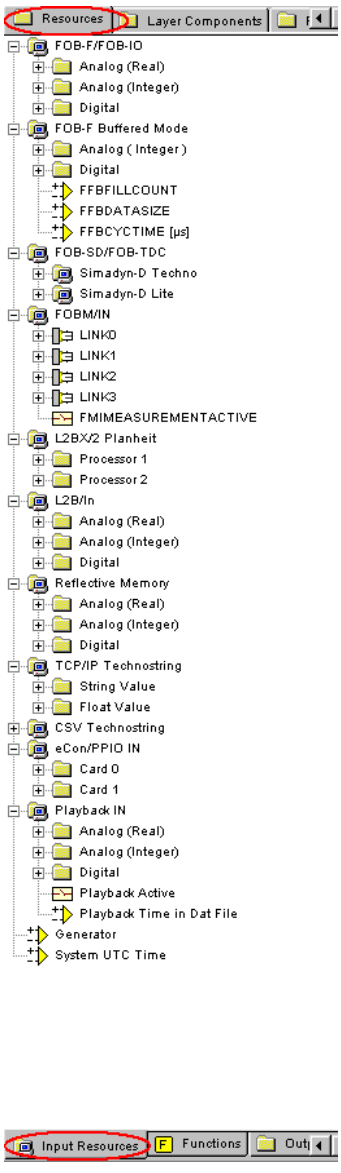
5 Process interface

The I/O process interface and the open communication interface of ibaLogic is based on the use of preconfigured and easy connectable input- and output resources. The available resources are shown in the resource area of the screen (tab "Resources"). By means of the resource selection tabs at the bottom choose between input- and output resources.

5.1 Input resources

The input resources are subdivided into the following groups::

Overview input resources



- FOB-F / FOB-IO (incl. FOB 4i PCI card)

Standardized analog and digital inputs, 32 groups (modules) with 32 inputs each (max. 1024). Incoming connection by fibre optical link from

1) PADU (Parallel Analog Digital Units)
2) ibaNet750 (WAGO) Remote-I/O-terminals or
3) SM64 / SM128V-cards.

With a PCMCIA-F card only the first two modules will be used.

- FOB-F Buffered Mode

These inputs refer to the first eight modules of a FOB-F card, buffered by ibaLogic environment.

Predefined set of input variables for measuring systems that use buffered measured values from FOB-F cards (e.g. for FFT and recording applications). Max. buffer depth is 256 values for up to eight modules with 32 channels each (8* 32 = 256 channels).

- FOB-SD card

Full automatic interface to SIMADYN-D or SIEMENS TDC control devices (CS12/13/14); it supports passiv and request mode.

1) SIMADYN-D Techno; predefined TechnoString.
2) SIMADYN-D Lite; predefined set of input variables by CS22

- FOB-M/IN

Predefined set of input variables for 25 kHz-measuring system with FOB-M / Padu8 ICP / Padu8 M (vibration monitoring)

- L2BX/2 Flatness

Predefined set of input variables for flatness measurement; connection by Profibus L2Bx-F or L2B x/8 PCI.

- L2B/In

Standardized analog and digital inputs, 32 groups (modules) with 32 inputs each (max. 1024). Incoming connection by profibus link from

1) S7 (only 28 Real Values per Module due to S7 limitations)
2) Any other Profibus Master

- Reflective Memory

Predefined set of input variables for a Reflective Memory connection. Analog (integer or real) and digital inputs devided in groups of 32 modules with 32 inputs each (max. 1024). Special hardware components (cards from VMIC) are required.

cont'd next page

cont'd input resources

- TCP/IP TechnoString

TCP/IP-input variables, one group of 16 STRING and one group of 96 FLOAT variables; assignment of variables to TechoString is done under menu ↪ *TechnoString* ↪ *TCP/IP...*

- CSV TechnoString

Choice of 128 TCP/IP input STRING variables; the single variable in the CSV-string is separated by comma (CSV = **C**omma **S**eparated **V**alue)

- eCon/PPIO IN

Predefined set of 32 input variables connected via the parallel port of the PC (printer port, lptx).

- PlaybackIn

Predefined set of analog and digital input variables to be supplied with data by iba data file in playback mode. 32 modules with 32 analog inputs (integer or real) and 32 digital inputs each.

- Generator

Signal generator for sine, rectangular, triangular or custom- shaped signal with easy parameterization.

- System UTC Time

System time to be connected and used with time controlled functions for display or evaluation.

5

5.1.1. FOB-F, FOB-IO or FOB 4i- Input Resources

The FOB-F, FOB-IO and FOB 4i – input resources are devided into groups of:

❑ Analog (real) Modules 1..32 or alternatively

❑ Analog (integer) Modules 1..32 and

❑ Digital Modules 1..32

Each module consists of 32 inputs, i.e. a maximum of 32 * 32 = 1024 analog and 1024 digital inputs are available.

Each fibre-optical connection of a FOB-F, FOB-IO or FOB 4i-card is linked to two modules with 32 inputs, i.e. a total of 64 analog and 64 digital inputs.

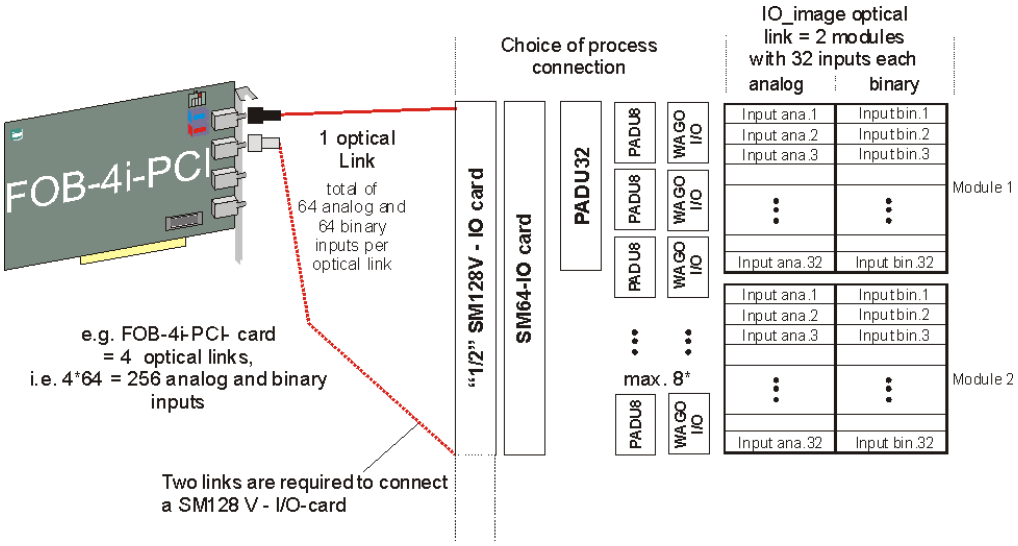


Fig. 70 FOB 4i PCI-card, FO-connectors

One optical link can be connected to:

- ❑ one SM 64-IO-card (64 analog and 64 digital signals)
- ❑ two PADU 32 devices (2*32 = 64 analog and 64 digital signals)
- ❑ eight PADU8-devices (8*8 = 64 analog and 64 digital signals)
- ❑ eight WAGO-terminal heads (8*8 = 64 analog and 64 digital signals)

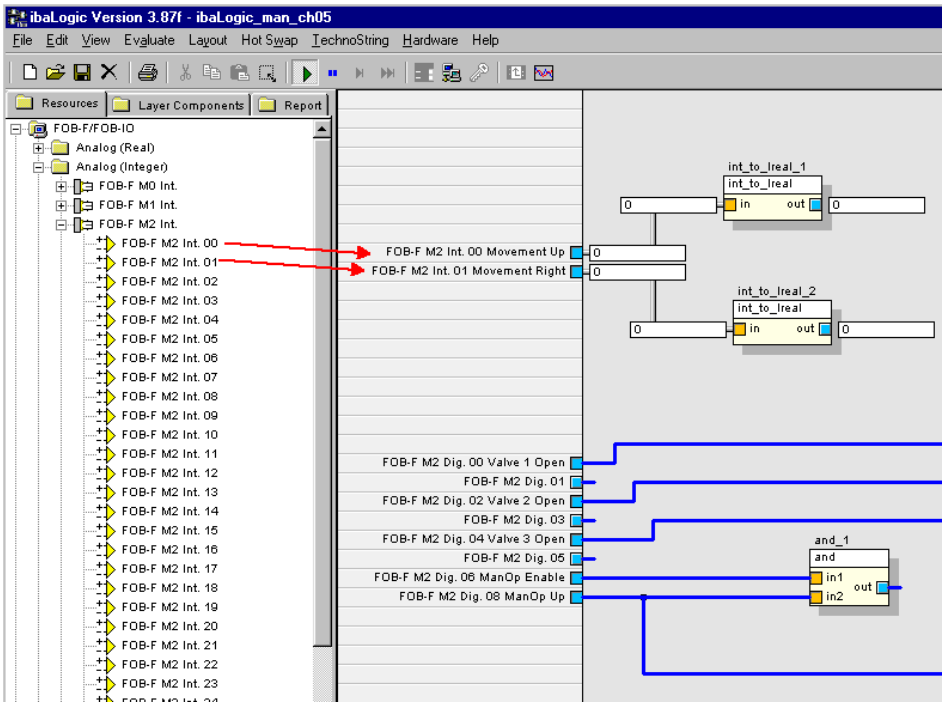


Fig. 71 FOB-F / FOB-IO input resources, placement in layout

The example in Fig. 71 shows the connection between ibaLogic and analog and digital FOB-F / FOB-IO - input resources.

It is not necessary to connect all resources of a module with one ibaLogic-task. Each signal can be selected individually and can be placed on the input signal margin, resp. on the output signal margin.

When needed, all inputs (resp. outputs) of a module can be placed on the input signal margin, resp. output signal margin by selecting the desired module and dragging it on the corresponding margin. The following query "Split array into single signals?" should be answered with "yes".

5.1.2. FOB-F Buffered Mode

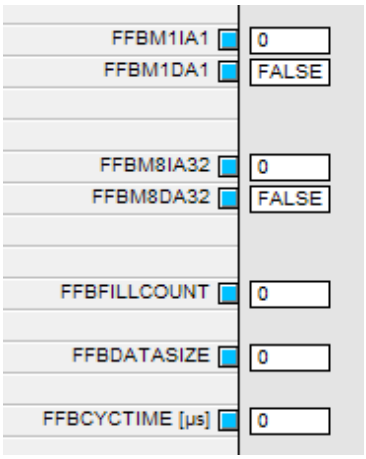
The group of "FOB-F buffered mode" input resources had been invented in order to process signals of a much higher sampling rate, acquired by the FOB-F card, than the sample time of a task in ibalogic would permit in continuous mode.

As an example 128 measured values (samples) of a signal which are required to evaluate a FFT can be processed even when the sample time of the FOB-F card for the data acquisition is about 1 ms but the sample time of the task is 50 ms

This has been made possible by a special measuring mode of ibaLogic, where data get buffered by the runtime environment and made available as arrays of a maximum depth of 256 values for the input resources. In order to prevent loss of samples the sampling rate of the task, i.e. of the ibaLogic layout, must be higher than the filling rate of the arrays.

For a reasonable use of this mode of operation select the ibaLogic *SignalManager mode*.

There may be other applications which require less than 256 samples or which don't need always buffered values or not all buffered values all the time. For these cases there is a special communication interface between the task and the ibaLogic runtime environment which provides the following inputs:



8 modules with 32 analog inputs (integer) each

8 modules with 32 digital inputs (bool) each

Fillcount is a counter to be increased by 1 everytime the buffer got filled up and the new buffered data had been transferred to the task.

Datasize is the actual number of samples which had been buffered.

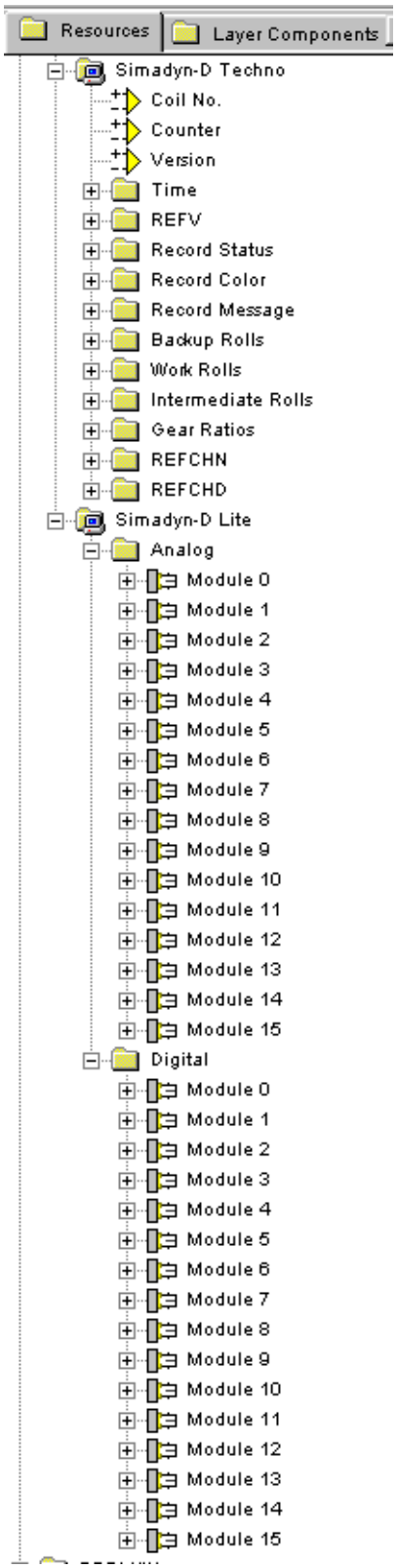
Cyctime is the actual sample time which had been used at the fiber optical link. This input is relevant for the so-called *asynchron mode*.

Fig. 72 FOB-F buffered mode input resources

5.1.3. Signals from Simadyn-D and TDC(FOB-SD / FOB-TDC)

Two types of signals are distinguished in case of a SIMADYN-D process interface:

- ❑ SIMADYN-D Techno (short for TechnoString)
- ❑ SIMADYN-D Lite (16 Modules, each with 32 analog (real) and 32 digital signals)



SIMADYN-D TechnoString (for FOB SD / FOB TDC)

The Simadyn-D technostring which is transmitted through the FOB SD supports the functionality and structure which is programmed within the Simadyn-D PLC only. This telegram provides all the necessary data to configure the QDA settings (i.e. FFT settings, stand settings, roll diameters etc.) for a 7 stand aluminum or steel mill. The structure is "hardwired" an cannot be changed. Data will be exchanged by a FOB-SD or FOB-TDC linking.

The connected Simadyn-D must provide a channel (type Refresh) with the name Q1DAT and a length of 512 Bytes exactly. For further explanations and comments which signals are used in which ranges please refer the respective Simadyn-D documentation.

Note: The Q2DAT channel (1920 Bytes) is no longer needed. This channel is replaced by the more practical MxPDADAT channels (see next chapter).

Q1DAT_AcqLength = 512 // Technostring channel must have 512 bytes!
Q2DAT_AcqLength = 0 // old data channel, no longer needed

SIMADYN-D Lite (for FOB SD / FOB TDC)

This resource set is structured very similar to FOB resources. A set of 8 analog and 8 digital "modules" with 32 channels each is provided. Each module can (but must not) be sent by one Simadyn-D CPU.

Note: FOB-SD have different resource types in ibaLogic. For CS22 use the Simadyn-D-Lite resources for FOB-SD the FOB-SD resource set!

In the Simadyn D / Simatic TDC PLC the data channels to be implemented must be named M0PDADAT to M7PDADAT with 132 Bytes length each (Type Refresh). Each channel represents one "module".

Some additional information for correct communication abilities are needed, especially the identifiers for the channel routing of Simadyn-D. Please refer to SIMADYN-D documentation.

For setup of FOB-SD and FOB-TDC there is a dedicated dialog under the menu ↪*File* ↪*PCI-Configuration* ↪*FOB-SD/TDC Settings*.

Please check also the iba_drv.cfg file for correct parameterization:

(//comments not to be found in the original file just added to explain the *.cfg structure contents).

"CS22.." means CS22 or FOB SD or FOB TDC!

```
FOBSX_AcqAddress = 0xE0000          // FOB SD base address
CS22_BgtName = PDA001               // name of SD-rack, see “struc” schematics for correct id
CS22_AcqAddress = 0xD0000           // always !!
Simadyn_Sync_Timeout = 15           // timeout here is 15 seconds
Simadyn_Proc_Timeout = 15           //
CS22_0_OwnName = DPDA1A             // a name of your free choice to baptize the “PC”
CS22_0_Partner = D1700B             // Coupling partner Dxx00B, where xx indicates
CS22_0_SoftwareVersion = V420       // where the CS1x motherboard is located; here slot 17
CS22_1_OwnName = DPDA2A             // This is the CS22 with the hw-id 01 and the name DPDA2A
CS22_1_Partner = D0900B             // which is plugged in slot 09 in the rack PDA001
CS22_1_SoftwareVersion = V430
CS22_2_OwnName = DPDA3A
CS22_2_Partner = D1200B
CS22_2_SoftwareVersion = V430       // version of the graphic design software “struc”
CS22_3_OwnName = DPDA4A             // the connected SD-CPU was structured with; here V4.30h
CS22_3_Partner = D1500B             // V4.25 must be parameterized with V4.20
CS22_3_SoftwareVersion = V430
CS22_NBoards = 1                    // number of active CS22 boards (not FOB-SD´s!)
Q1DAT_AcqLength = 512               // always when using a technostring
Q2DAT_AcqLength = 0                 // always !!
M0DAT_AcqLength = 132               // Note all channels have fixed structure and length
M1DAT_AcqLength = 0                 // shorter channels must be filled up with zeroes
M2DAT_AcqLength = 0                 // For every “module” with 32 analog plus 32 binary
M3DAT_AcqLength = 0                 // values a channel of 132 bytes length is needed
M4DAT_AcqLength =0                  // M0DAT corresponds to module1, M7DAT to module8
M5DAT_AcqLength =0
M6DAT_AcqLength =0
M7DAT_AcqLength =0
```

5.1.4. Input Resources FOB-M/IN

FOB-M process interfaces are used in conjunction with Padu8-M, resp. Padu8-ICP, analog-digital converters with a sampling rate of 40 µs (25 kHz) for the purpose of vibration monitoring of machines. The following table shows the configuration of channel 1 (link1) of the first FOB-M module. Up to four channels are possible.

FMIL1PADUNUM	1
FMIL1SAMPLETIME	2000.0
FMIL1GAIN1	0.0
FMIL1GAIN2	0.0
FMIL1GAIN3	0.0
FMIL1GAIN4	0.0
FMIL1GAIN5	0.0
FMIL1GAIN6	0.0
FMIL1GAIN7	0.0
FMIL1GAIN8	0.0
FMIL1FREQ1	0.0
FMIL1FREQ2	0.0
FMIL1FREQ3	0.0
FMIL1FREQ4	0.0
FMIL1FREQ5	0.0
FMIL1FREQ6	0.0
FMIL1FREQ7	0.0
FMIL1FREQ8	0.0
FMIL1RCMD	-254
FMIL1CMD	-3580
FMIL1ANA_DATA1	0
FMIL1ANA_DATA2	0
FMIL1ANA_DATA3	0
FMIL1ANA_DATA4	0
FMIL1ANA_DATA5	0
FMIL1ANA_DATA6	0
FMIL1ANA_DATA7	0
FMIL1ANA_DATA8	0
FMIL1Dig_DATA1	FALSE
FMIL1Dig_DATA2	FALSE
FMIL1Dig_DATA3	FALSE
FMIL1Dig_DATA4	FALSE
FMIL1Dig_DATA5	FALSE
FMIL1Dig_DATA6	FALSE
FMIL1Dig_DATA7	FALSE
FMIL1Dig_DATA8	FALSE
FMIL1DATAAVAILABLE	FALSE
FMIL1DATASIZE	0
FMIL1LINKAVAILABLE	TRUE
FMIL1LINKMEASURING	FALSE
FMIL1DATALOST	FALSE
FMIL1DATAOVERRUN	FALSE

Number of active Padu8-ICP unit (00 ... 96)
Sample time in µs for this Padu8-ICP unit

\
| Actual gain setting for channels 1...8, given in dB
| The current setting of each channel is indicated.
|
/

\
|
| Actual setting of corner frequency (Hz) for channels 1...8
| The current setting of each channel is indicated.
/

Current state of "reset" command
Current processing command

\
| Analog input channels 1...8 (signed INTEGER)
|
|
/

\
|
| Binary input channels 1...8 (BOOL)
|
/

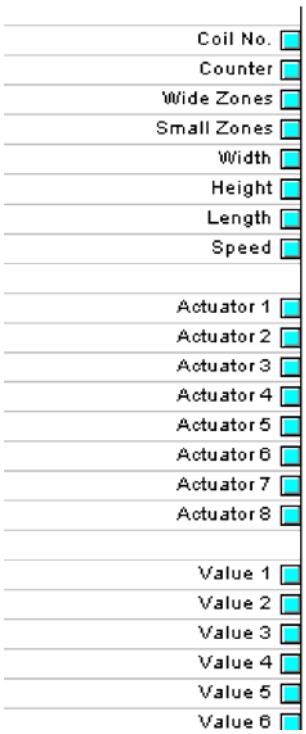
- State of input buffer; TRUE = number of values exceeds buffer
- Size of data (multiples of 10), as soon as data are available
- State of link; TRUE, if link is ok
- TRUE if link state ok and Padu is activated for measuring (FOMEASUREMENTSTART = TRUE)
- DATALOST = TRUE, if data rush in faster than beeing processed
- OVERRUN = TRUE, if buffer overflow and measurement interrupted

FMIMEASUREMENTACTIVE	FALSE

TRUE, if FOB-M measurement is running

5.1.5. L2Bx/2 Flatness

This specialized input resource was developed for the connection between ibaLogic and a SIEMENS flatness control. The link between the two systems is a Profibus L2-DP with the flatness-PC as Profibus master and ibaLogic (FOB L2B-card) as slave. In order to start a communication both master- and slave address must be known and configured. The FOB L2B-card should be parameterized in one of the flatness modes (see below). No matter which mode is selected, the incoming data will always be assigned to the same ibaLogic input resources.



The dataset to be transmitted comes with header information (Coil No., Counter, Zone Width etc.) in order to control the QDA-display.

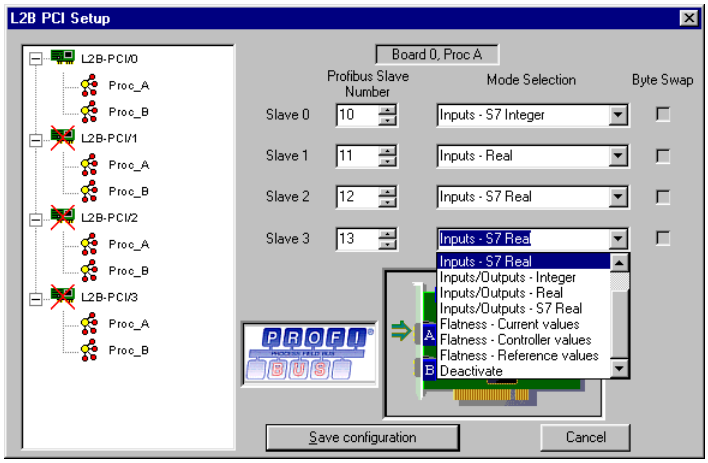
Beside of eight actuators there are up to 80 zone values.

On the FOB L2B-card, two input resources (processor 1 and 2) are available for connection of up to two flatness control PCs.

ibaLogic monitors the Profibus-link. An interrupted connection will be detected and reinstalled automatically. When "offline" (interrupted), ibaLogic freezes and keeps the most recent received data. In this case the QDA-flatness profile shows no further alteration of values.

Note: An interruption of the Profibus-link will not affect the time behaviour of ibaLogic.

L2B – card configuration



When establishing a connection between ibaLogic and the target system, only the data with reference to the selected mode will be requested. The target system will adjust itself in compliance to the selected mode. An alternation of the mode during operation is not permitted.

↗ *see also chapter 2.6.3*

5.1.6. Reflective Memory (RM)

The linking of RM-resources and RM-interface is part of the PCI configuration as described in chapter 2.6.5

Each of the 32 RM-input modules consist of 32 input signals whose signal names are clearly assigned to the modules. Additionally, each signal has a description (text) which can be edited in order to improve the technical comprehension by the user.

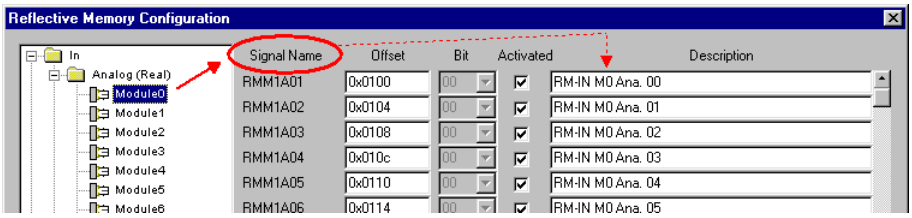


Fig. 73 Reflective Memory input resources, connection between module, signal name and description

The descriptions of the input signals appear also in the resource tree and further in the layout when the signals are used. They also can be found in the tooltip when placing the mouse cursor over a corresponding connector.

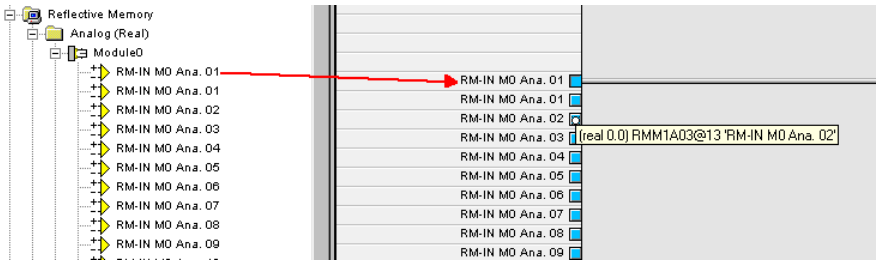


Fig. 74 Reflective Memory input resources, appearances of signal description

5.1.7. TCP/IP-TechnoString

The TCP/IP TechnoString functionality is always defined as a certain structure between two partners. Any kind of data can be transmitted (float values, strings etc.). This type of technostring needs a hard structure in means of how long (how many bytes) a specific parameter or part of the technostring is. The assignment is done with the help of the menu ↪*TechnoString* ↪*TCP/IP...* of ibaLogic. Any part of the TechnoString can be selected and assigned to a TCP/IP-String variable (1...16).

As a precondition for using this functionality the TCP/IP communication must have been activated in the menu ↪*File* ↪*System settings* ↪*Other*. The checkbox *TCP/IP Activate* must be checked off.

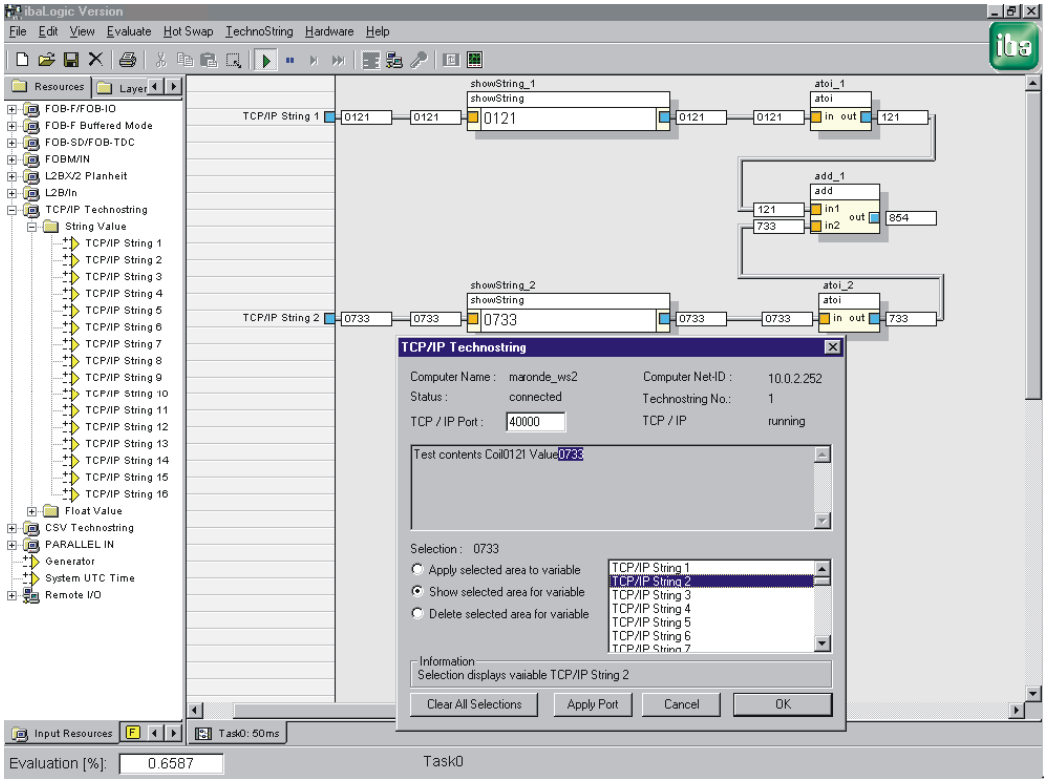


Fig. 75 Example: Assignment of TCP/IP-String 2 to selected parts of the received TechnoString

The example in Fig. 75 shows how a selected part of the TechnoString (here: characters "0733") is assigned to the variable "TCP/IP String 2". In order to do so, please follow these steps:

1. Choose menu ↪*TechnoString* ↪*TCP/IP*.
2. In the field *TCP/IP Port* please enter the same port number which is used by the source system (sender) for this TCP/IP communication.
3. In order to check the communication the source system may send a sample string message or you should use the software tool *TcpIpTest...exe* from iba in order to create a sample string and send it to ibaLogic. In any case the sample string should appear in the dialog *TCP/IP Technostring*.
4. Check the option *Apply selected area to variable*.
5. With the mouse cursor mark the characters in the displayed TechnoString which should be assigned to a TCP/IP String variable. (If marking is not possible please make sure that no technostring is beeing sent at this time.)

6 Click on the desired variable in the list of variables (here: TCP/IP String 2) which should be connected to the marked part of the string. Ready!

In this way all TCP/IP String variables may be assigned to different parts of the TechnoString.

It is essential that the TechnoString has a fixed structure, i.e. the same data must always be at the same place inside the string. If, in the example above, "Value733" would be sent instead of "Value0733" all following characters would be shifted by one position to the left and TCP/IP String variables referring to these following characters wouldn't have the correct value. As a consequence, leading zeros should be used, if applicable.

For the purpose of TechnoString reception only the above mentioned settings are required. The settings concerning TCP/IP and TechnoString in the menu ↪File ↪PCI-Configuration ↪ TCP/IP Out Settings have nothing to do with the reception of TechnoStrings. These settings only refer to the output or sending of TechnoStrings. (see also chapter 5.2.5)

5.1.8. CSV-TechnoString

The CSV TechnoString is another method to transmit data to ibaLogic. All values should be separated by commas (CSV = **C**omma **S**eparated **V**alues). Due to the commas as separating signs, no fixed format of strings and values is required and so it's somehow easier and more flexible than the TCP/IP-TechnoString method. The fields of characters can be generated by MS-Excel or other programs which are able to create files with comma separated values.

ibaLogic receives the data as a chain of characters (fields) and assigns them automatically to the CSV-String 1...128. The assignment occurs according to the order of the source definition.

The source should have the following format:

< field1>,< field2>,.....,.....,< field128> < cr > < lf >

Example:

Create a text file named "pipetest.txt" with a contents as follows (4 fields):

```
CSV-Test,1234,5678,hallo < cr, lf >
```

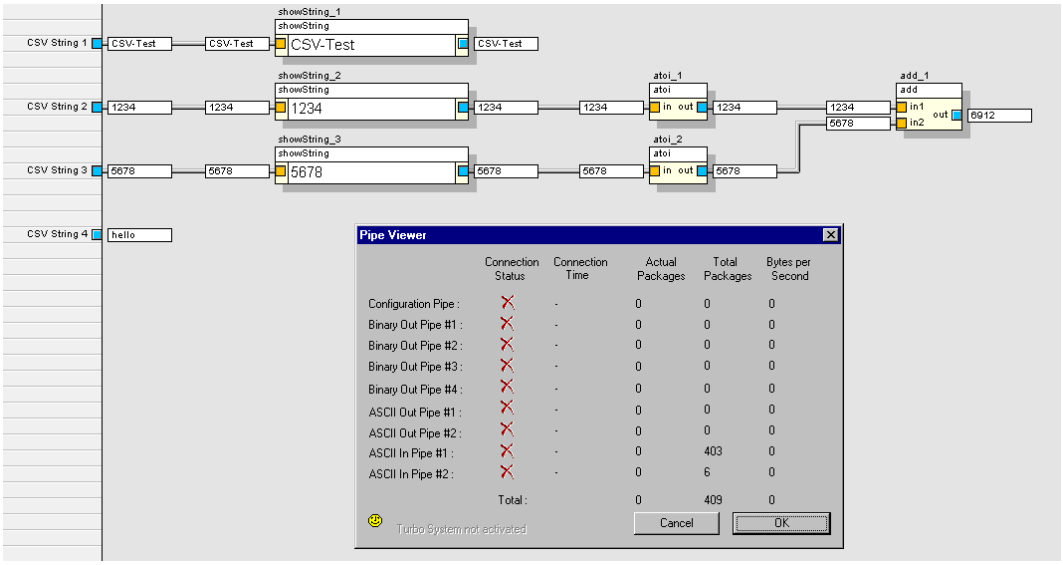
Don't forget to add the "carriage return" and the "line feed" at the end of the file.

Forward the file to the receiving PC, named "PDA", by using the DOS-command

```
copy pipetest.txt \\PDA\pipe\qda_asciiin
```

"qda_asciiin" is the keyword for the ibaLogic-Pipe (the three "i"s are correct!).

ibaLogic receives the data as a chain of characters and provides them as input variables "CSV String 1...128" for further use. The conversion into other data types is done by converting function blocks, e.g. ASCII to integer.



The state of "ASCII In Pipe #1 and #2" can be monitored by using the menu ↪ *View* ↪ *Pipes...*

5.1.9. eCon/PPIO IN – inputs from eCon / eCon32

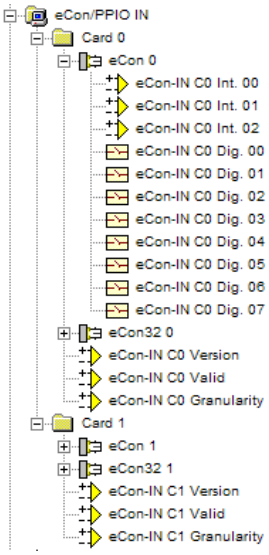
These input resources are dedicated to the eCon and eCon32 devices from iba.

The eCon devices are small I/O devices which have to be connected to a PC via the parallel printer port. There are two types available:

eCon: This type consists of 3 analog inputs (AI), 2 analog outputs (AO), 8 digital inputs (DI) and 8 digital outputs (DO).

eCon32: This device provides 32 digital inputs and 32 digital outputs.

Up to two of these devices can be operated in combination by one parallel PC port.



The assignment of eCon devices and input resources is as follows:

Card 0	first eCon at parall port if eCon, then 3 AI and 8 DI if eCon32, then 32 DI
Card 1	second eCon, connected in line to the first eCon if eCon, then 3 AI and 8 DI if eCon32, then 32 DI

The input signals ...*Version*, ...*Valid* and ...*Granularity* provide information about the connected device:

Version:	Firmware version of the device,
Valid:	Status indication whether input values are valid or not,
Granularity:	Step width depending on A/D converter resolution. A 10 bit converter resolution leads to a step width of 64.

For further informationen concerning the eCon devices please refer to the related hardware documentation. That documentation also cares about the software engineering.

hw_man_econ_en_A4.pdf

5.1.10. PlaybackIN – inputs for the playback operation mode

The input resources PlaybackIn had been invented especially for the operation with iba data files (dat-files) as signal source. They have to be configured by module assignment under menu ↪*File* ↪*Program Settings* ↪*Playback* ↪*Module Assignment.*

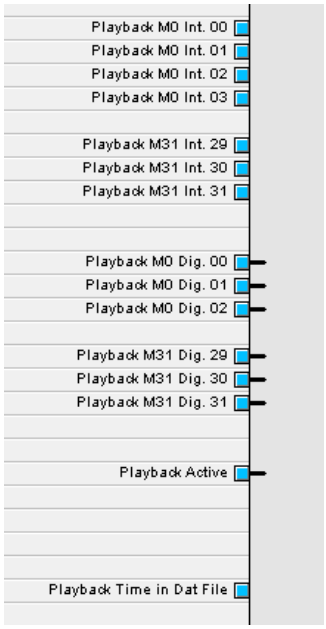
↗ *See also chapter 2.4.4* and 3.6.4

Depending on the data type of the values as they are available in the dat-file, the datatype of the input resources (integer or real) will adjust automatically.

The signal names will NOT be taken from the dat-file. They have to be entered manually, if necessary.

A quantity of 32 * 32 input signals are provided in order to read dat-files of an extended ibaPDA-system with 1024 analog and 1024 digital signals.

Using the optional operation mode *with hardware I/O* (menu ↪*File* ↪*System settings* ↪*Other*, *Playback settings)* it's even possible to combine the playback inputs with real online inputs over FOB- or L2B-cards.



32 modules with 32 analog values each (integer or real)

32 modules with 32 digital values each

Playback Active is = TRUE, if the playback mode is active (menu ↪File ↪System Settings ↪General).

Playback Time in Dat File returns the current position of the "cursor" in the dat-file. This value is given in seconds, relativ to the start date of the recording in the dat-file.

5.1.11. Generator

The input resource Generator is a practical tool. It's an easy way to generate test signals of different wave forms.

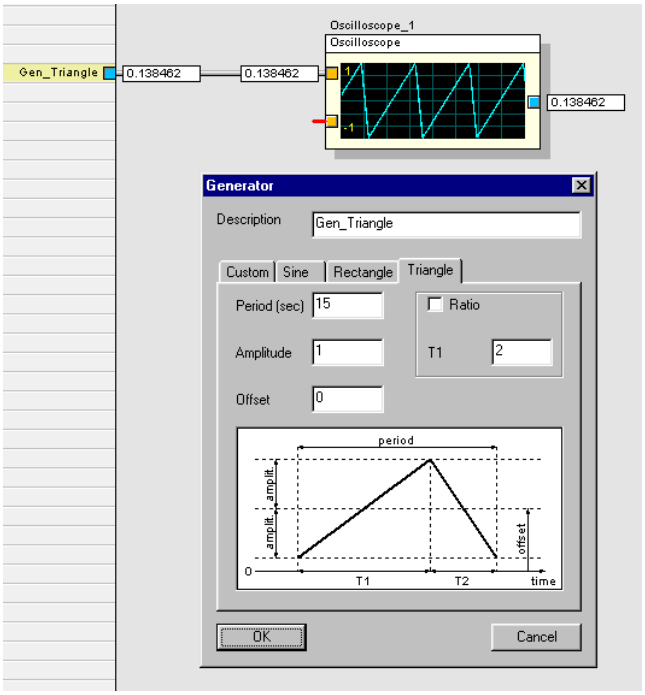


Fig. 76 Input resources, Generator

In order to use a generator signal just select the input resource Generator and drag it to the input margin of the layout. As many instances of the genarator as needed may be used with different wave forms at a time.

After drag&drop of the generator input a dialog opens as shown in Fig. 76 and the following settings can be made:

❑ *Description*

This text entry will appear as name of the generator signal in the layout and should decribe the signal clearly. Particularly when using many generator signals this helps keeping clarity.

❑ *Tabs with generator types*

Under each tab there is a diagram which shows the characteristics of the corresponding wave form.

All generator types have the following parameters in common:

- *Period*: Entry of the time of a full period, in sec.
- *Amplitude*: Amplitude of the signal; there is only one value, taken for both positive and negative amplitude.
- *Offset*: Entry of offset (X-axis); if the signal should always be positive, the offset must have the same value as the amplitude.

Moreover, there are other generator-specific parameters:

❑ ***Tab Custom***

This generator type allows the customized definition of a periodic signal. The period will always be devided in 20 even parts (index). For each index (1...20) a single value may be entered. In order to ease the work it's possible to choose one of the other generator types first (Sine, Rectangle, Triangle) and then switching back to the tab Custom. The wave form of the previous generator type is now the basis for the customized generator and the values can be adjusted easily. The value adjustment can be performed by entering values in the index-related field or by using the mouse on the curve in the diagram.

❑ ***Tab Sine***

The sine signal doesn't require further settings.

❑ ***Tab Rectangle***

A rectangle signal can be asymmetric in temporal terms. The total duration of a period is defined by the parameter *Period*. The two parts of a period can be adjusted by the parameter *T1* (given in sec.). If the option *Ratio* is checked, the value in the field *T1* is the ratio of T1/T2.

❑ ***Tab Triangle***

The same remarks as for rectangle apply correspondingly.

5.1.12. System UTC Time

ibaLogic works on a so called realtime base, i.e. actions can be triggered by date and time in ibaLogic. For that, the resource *System UTC Time* and function blocks, such as *SplitUtcTime*, are provided.

Sometimes problems may occur during switch-over from or to daylight saving time because it depends on how and when the system was configured.

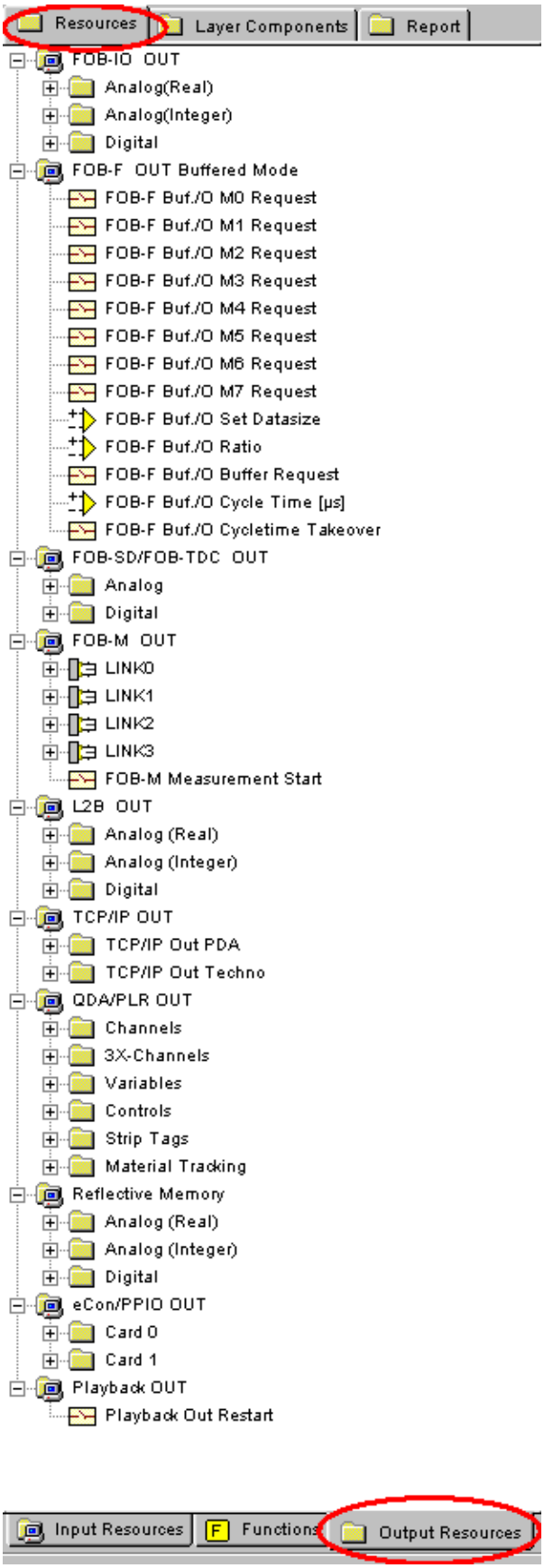
***Note**: Daylight Saving Time:*

*In the properties of Date/Time-settings in Windows® NT (*Start ↪ Settings ↪ Control Panel ↪ Date/Time*) you should uncheck the option "automatic daylight saving time".*

This has to be done prior to a change to daylight saving time. Otherwise it's useless.

5.2 Output Resources

The output resources are devided into the following groups:



- FOB-IO/ OUT

Standardized analog and digital outputs, 32 groups (modules) with 32 outputs each (max. 1024). Outgoing connection by fibre optical link to:

1) PADU (Parallel Analog Digital Units)
2) WAGO Remote-I/O terminals or
3) SM64- / SM128V-cards

- FOB-F/ OUT Buffered Mode

Predefined set of output variables for control of measuring systems that use buffered measured values from FOB-F cards (e.g. FFT applications).

Individual data request for up to eight modules.

- FOB-SD / FOB-TDC OUT

Full automatic interface to SIMADYN-D or Simatic TDC control devices (CS12/13/14 or GDM); eight groups (modules) with 32 outputs each for analog and digital outputs (max 256).

- FOB-M/ OUT

Predefined set of output variables for 25 kHz-measuring system with FOB-M / Padu8 ICP (vibration monitoring)

- L2B/ OUT

Standardized analog and digital outputs, 32 groups (modules) with 32 outputs each (max. 1024). Outgoing connection by Profibus network to:

1) Profibus Slave (e.g. Simatic S7)

- TCP/IP-OUT

TCP/IP output variables, groups of

1) TCP/IP outputs to PDA-system, 16 modules with 32 analog and digital channels each (max. 512)

2) TCP/IP outputs for TechnoStrings, four output strings with data and four control outputs

For output status see menu ↪ *View* ↪ *TCPIP Out...*

- QDA/PLR-OUT

Predefined set of output variables to QDA- or PLR-system.

- Reflective Memory

Predefined set of output variables for Reflective Memory (RM) connection; 32 groups (modules) of 32 analog (integer or real) and 32 digital outputs each (max. 1024). The RM connection requires special hardware components / interface cards.

- eCon/PPIO OUT

Predefined set of 32 output variables to the parallel printer port of the PC.

- Playback OUT

One digital "output" for restart of playback.

5.2.1. FOB-IO or FOB 4o-Output Resources

The FOB-IO output resources are devided into groups of

- ❑ Analog (real) modules 0...31 or
- ❑ Analog (integer) modules 0..31 and
- ❑ Digital modules 0...31

Each module consists of 32 outputs, i.e. a maximum of 32 * 32 = 1024 analog and 1024 digital outputs are available.

Example below: Each fibre-optical connection of a FOB-IO or FOB 4o-card is linked to two modules with 32 inputs, i.e. a total of 64 analog and 64 digital outputs.

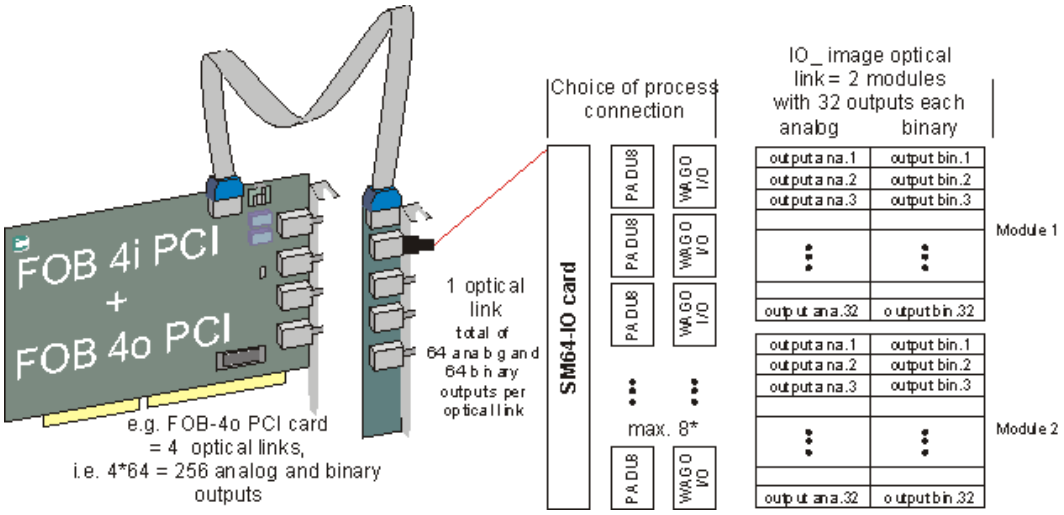


Fig. 77 FOB 4o, output connections

One optical link can be connected to:

- ❑ one SM 64-IO-card (64 analog and 64 digital signals)
- ❑ eight PADU8-output devices (8*8 = 64 analog and 64 digital signals)
- ❑ eight WAGO-terminal heads (8*8 = 64 analog and 64 digital signals)

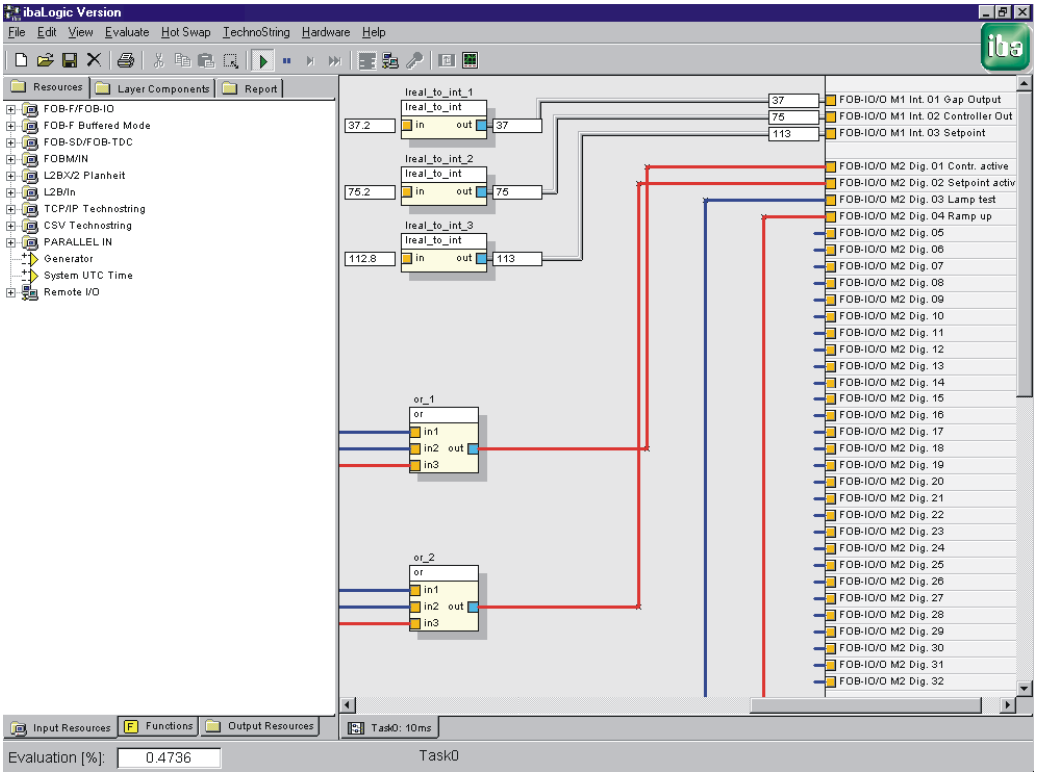


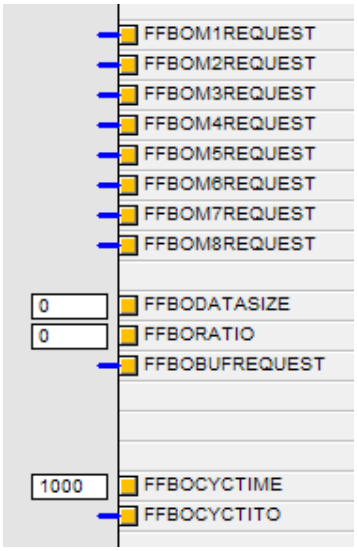
Fig. 78 FOB-IO output signals, example

The example in Fig. 78 shows the connection between ibaLogic and analog and digital FOB-IO - output resources.

When needed, all outputs of a module can be placed on the output signal margin by selecting the desired module and dragging it on the corresponding margin. The following query "Split array into single signals?" should be answered with "yes".

5.2.2. FOB-F OUT Buffered Mode

These output resources are dedicated to the FOB-F buffered mode and are used only for contol of the reading of the buffered inputs. These are no data outputs to an external process. (see also chapter 5.1.2)



8 digital outputs for a focused module-specific request of buffered data from the FOB-F interface (optimization of processor load and reduction of administrative tasks).

..Datasize is the quantity of measured values (samples) that should be provided at a time by the ibaLogic runtime environment (max. 256).

..Ratio is an integer multiple of the number of samples in a sample time. E.g., Ratio = 2 means, that only every second sample will be written to the buffer.

..Bufrequest is the control output to the ibaLogic runtime environment. *..Bufrequest* = TRUE means that the quantity of data with reference to *..Datasize* and *..Ratio* should be buffered. The buffer contents should then be transferred to the ibaLogic task and the input *FillCount* should be incremented by 1.

..Cyctime is the cycle time to be transmitted in asynchron mode at the fiber optical link (1 ...10 µs).

..Cyctito is the control signal (take-over) for the cycle time to be transmitted in asynchron mode.

5.2.3. FOB-SD / FOB-TDC OUT – Output Resources

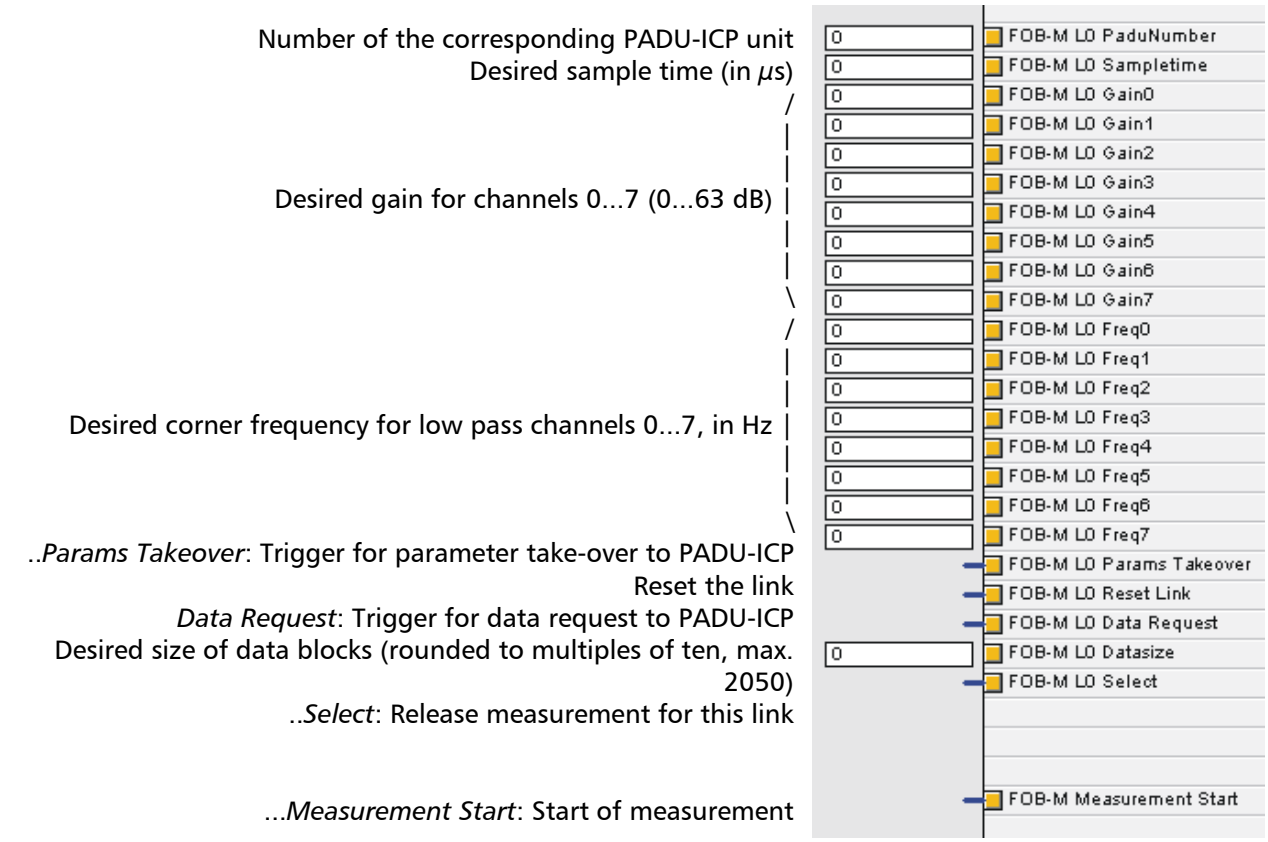
The outputs are part of the full automatic interface to SIMADYN-D or Simatic TDC control devices. Like for the FOB-IO interface card, the output resources for FOB-SD / FOB-TDC are devided into groups of

- ❑ Analog (real) modules 0...7 and
- ❑ Digital modules 0...7

Each module consists of 32 outputs, i.e. a maximum of 8 * 32 = 256 analog and 256 digital outputs are available.

5.2.4. FOB-M /Out – output resources

The FOB-M output resources are used to activate and to parameterize the PADU-ICP unit (25 kHz measurement). Up to four links to a PADU-ICP (eight channels each) are supported by ibaLogic (two FOB-M with two links each).



For changing parameters the running measurement has to be stopped. Then the parameters can be transmitted to the PADU-ICP.

The PADU-ICP unit needs approximately 10 sec for internal evaluation of a new gain. After the parameterization is finished the unit sends the new data continuously to ibaLogic. The process of parameterization may affect other I/O interfaces (e.g. FOB-IO) because the ibaLogic-I/O driver has to be stopped for two cycles!

Data buffer:

In order to guarantee a proper data transmission of continuous data blocks, different data buffers of fixed size are installed:

- ❑ FOB-M interface, buffer size: 1024 values per channel
- ❑ I/O driver, buffer size: 25.000 values per channel
- ❑ ibaLogic, buffer size: 50.000 values per channel

These figures lead to the resulting sample times, resp. task cycle times as follows:

- ❑ PADU-ICP sample time: e.g. 40 µs
- ❑ Size of data blocks: e.g. 2050 values
- ❑ ibaLogic task cycle time: e.g. 25 ms

1 / 25 ms * 2050 = 82.000 values/sec/channel Data Read Rate (DRR)

1 / 40 µs = 25.000 values/sec/channel Data Generation Rate (DGR)

Rule:

The data read rate should be at least three times the data generation rate!

A loss of one sample cycle must not cause a data loss in ibaLogic.

5.2.5. TCP/IP-Output Resources

The TCP/IP output resources are devided in two main groups:

- TCP/IP Out PDA, output of data for an ibaPDA-system
- TCP/IP Out Techno, output of TechnoStrings, e.g. to an ibaPDA-system

5.2.5.1. TCP/IP-Out PDA – signal outputs to a PDA-system

❑ Analog (real) modules 0...15,

❑ Digital modules 0...15 and

❑ Control control outputs, one per module 0...15

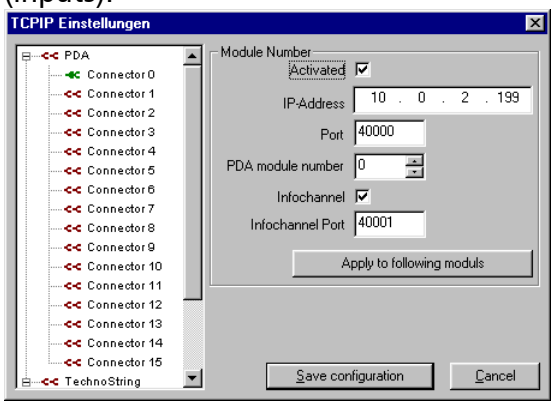
Each module consists of 32 outputs, i.e. a maximum of 16 * 32 = 512 analog and 512 digital outputs are available for transmission from ibaLogic to ibaPDA via TCP/IP.

For the purpose of transmission control there are 16 control outputs. The transmission of data can be controlled (start/stop) individually for each channel 0 ...15 (i.e. for modules 0...15). To enable the transmission of data the corresponding control output *TOUTPDA Send xx* must be set on TRUE.

Setup for data output to an ibaPDA-system

1 In menu ↪*File* ↪*System settings* ↪*Other* check off the TCP/IP activation checkbox.

2 In the same dialog click on the Configuration button (or alternatively over menu ↪*File* ↪*PCI Configuration* ↪*TCP/IP Out settings*) to open the dialog for the TCP/IP settings. The settings in this dialog only refer to the output of TCP/IP data. They are not relevant for TCP/IP reception (inputs).



3 Click on the first "Connector" in the tree just under the branch "PDA". Each connector corresponds exactly to one module in the *TCP/IP Out PDA* output resources.

4 Now activate this connector by checking off the checkbox in the right part of the dialog window. Enter IP-address of the target PC (ibaPDA-PC) and the mutual port number. Due to the individual addressing of the different connections it is possible to supply different ibaPDA-systems with data.

5 Furthermore, it's possible to reassign the output signals to other module numbers than 0...15 in the ibaPDA-system. This might be necessary when these module numbers are already occupied in the ibaPDA-system by other data sources (Padus etc.).

6 As an option the transmission of an infochannel can be enabled or disabled. The infochannel is used for transmission of additional information which can be found later in the dat-file.

7 If data of more than one module (connectors) should be transmitted to the same ibaPDA-System then click on the button *Apply to following modules*. The settings will be copied to the modules (connectors) below the current one.

8 Close the dialog by clicking on the button "Apply" or Save Configuration respectively.

An active connection is indicated by a green symbol.

5.2.5.2. TCP/IP Out Techno outputs

- ❑ Data (string) up to four TechnoStrings 0...3 and
- ❑ Control control outputs for each string

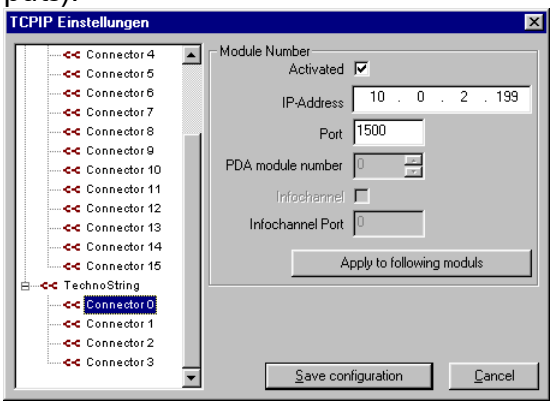
Each TechnoString output can contain ASCII-strings of up to 1024 characters, including termination (0 hex).

In order to control the TCP/IP transmission a group of control outputs is provided. Each of the communication channels 0...3 (corresponding to TechnoStrings 0...3) can be startet or stopped by these control outputs. The transmission of the strings is enabled when the corresponding control output *TOUTTECHNO Send x* is set on TRUE.

Setup for Technostring output

1 In menu ↪*File* ↪*System settings* ↪*Other* check off the TCP/IP activation checkbox.

2 In the same dialog click on the Configuration button (or alternatively over menu ↪*File* ↪*PCI Configuration* ↪*TCP/IP Out settings*) to open the dialog for the TCP/IP settings. The settings in this dialog only refer to the output of TCP/IP data. They are not relevant for TCP/IP reception (inputs).



3 Click on the first "Connector" in the tree just under the branch "TechnoString". Each connector corresponds exactly to one TechnoString, i.e. one *TCP/IP Out PDA* output resource.

4 Now activate this connector by checking off the checkbox in the right part of the dialog window. Enter IP-address of the target PC (e.g. ibaPDA-PC) and the mutual port number. This port number should differ from the port number for data transmission. Due to the individual addressing of the different connections it is possible to supply different ibaPDA-systems with TechnoStrings.

5 If more than one TechnoString (connectors) should be transmitted to the same ibaPDA-System then click on the button *Apply to following modules*. The settings will be copied to the modules (connectors) below the current one.

6 Close the dialog by clicking on the button "Apply" or Save Configuration respectively.

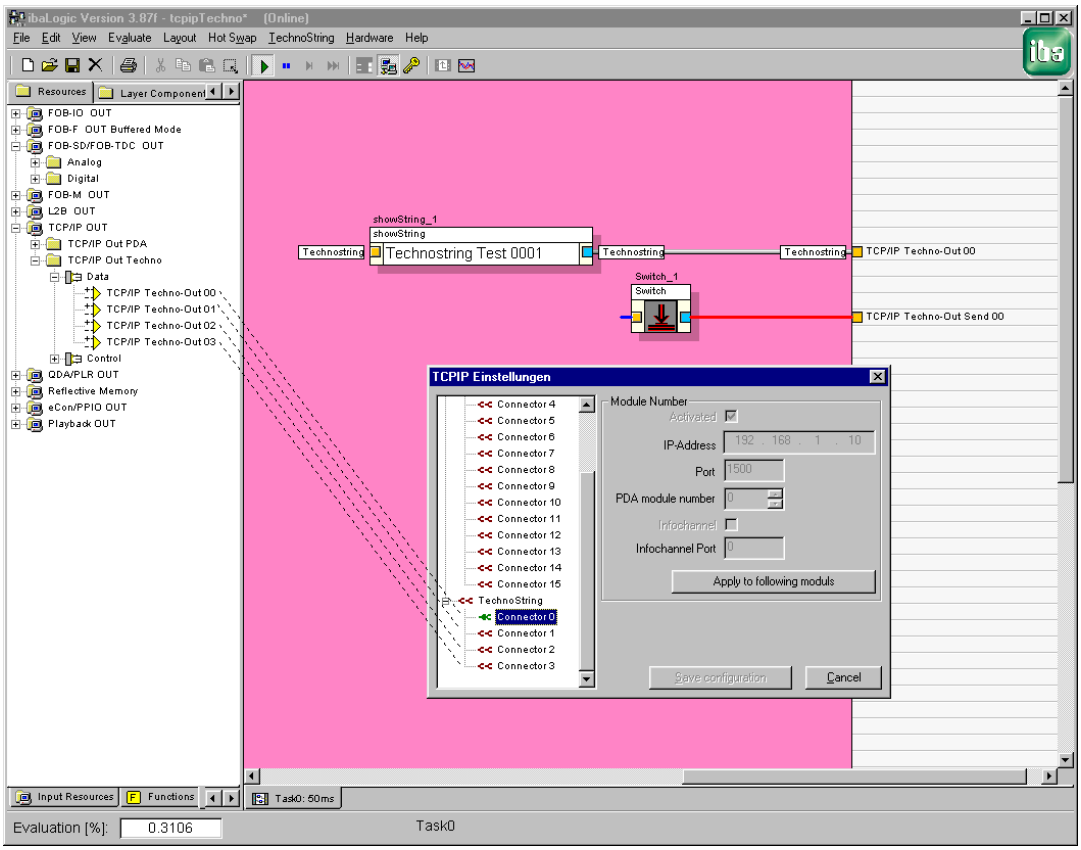


Fig. 79 TCPIP TECHNO Out, connection between output signals and TCPIP settings

ibaLogic always transmits the TechnoString with an empty termination (0 hex). Therefore, it is required to enter another termination (0) instead of carriage return in the TechnoString setup-dialog in ibaPDA.

Moreover, it is strongly recommende that no other user in the TCP/IP network uses the same port numbers which are used for the TechnoString communication. Otherwise, system interferences may occur.

Setup with older versions of ibaLogic:

ibaLogic versions < 3.83c provide the setup of TCP/IP communication in the ISA configuration dialog.

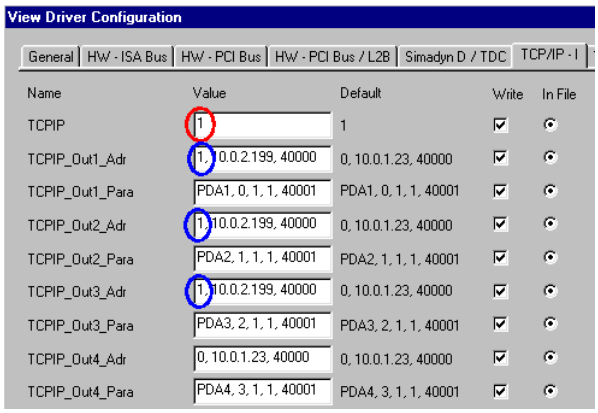


Fig. 80 TCP/IP setup in former ibaLogic versions

5

Before releasing the TCP/IP – PDA outputs the corresponding driver must be released ("TCPIP" = 1). Each output has to be configured with the following entries: enable output (1), IP-address of the ibaPDA-PC (e.g. 10.0.2.199) and port number (e.g. 40000).

The TCP/IP-**PDA** Out-channels should be configured with the parameters *TCPIP_Out**1**_Adr* / *..._Para* to *TCPIP_Out**16**_Adr* / *..._Para*.

The TCP/IP **Techno** Out-channels should be configured with the parameters *TCPIP_Out**17**_Adr* / *..._Para* to *TCPIP_Out**20**_Adr* / *..._Para*.

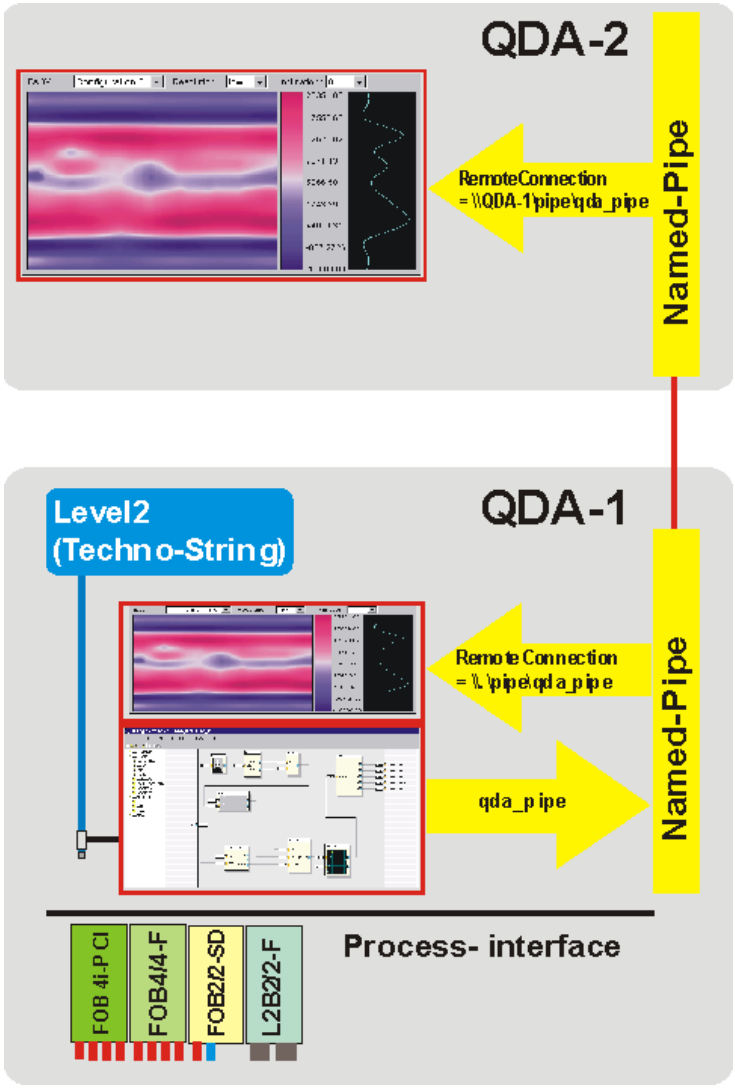
To get the configuration dialog, use menu ↪*View* ↪*ISA Configuration...* and enter the required information under the tabs "*TCP/IP-I*" and "*TCP/IP-II*". Save the configuration and restart ibaLogic.

This dialog is also still available in up-to-date versions of ibaLogic in the menu ↪*File*, but it's disabled when no ISA-card has been detected.

5.2.6. QDA Out- output resources

In order to understand the communication between QDA and ibaLogic, please follow this short introduction to the topic of "Named Pipes". QDA and ibaLogic use the "Named Pipes"-method for connection over TCP/IP networks.

Basic properties of ibaLogic's communication by "Named Pipes":



The use of Named Pipes offers the possibility to use multiple synchronized PC-workstations. As a benefit of this concept, the workstations can be placed whereever they are needed. Usually the first PC-workstation is placed in the switch-house or control room. This first workstation provides the necessary hardware components for the process interface and collects the data to be measured. More PC-workstations can be placed on control pulpits, maintenance stations or whereever it makes sense.

ibaLogic uses the "Named Pipes"-concept for communication with many other applications, even with itself when several ibaLogic-applications are running on different workstations. "Named Pipes" is a TCP/IP application layer functionality which is available on all Windows NT® workstations.

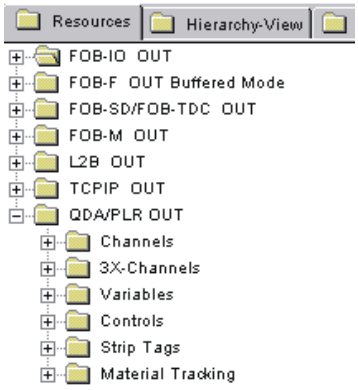
The example (left) shows two synchronized QDA-PC-workstations which are connected with a single ibaLogic source by Named Pipes. The entire set of data is sent both to the local QDA on PC "QDA-1" and to the second PC "QDA-2".

Remark: The reference "QDA-1" on the PC QDA-2 is a reference to the name of the PC and not to the ibaLogic application! "qda_pipe" is the address reference for the application.

As soon as a signal has been defined, all other applications which are following in terms of data flow are able to use the signal name immediately. So, every signal has to be named only once.

5.2.7. QDA/PLR OUT - resources

All control and data connections for QDA are managed in the QDA/PLR OUT section of the ibaLogic-output resources.



There are six groups of resources:

- Channels
- 3X-channels
- Variables
- Controls
- Strip Tags and
- Material Tracking

5.2.7.1. Channels

5

ibaLogic supports the use of up to 96 channels which are structured as follows:

- ❑ Value CH # (float) // Signal value
- ❑ Reference CH # (float) // Reference value
- ❑ Low. Limit CH # (float) // QDA lower signal limit value
- ❑ Up. Limit CH # (float) // QDA upper signal limit value

The channels can be selected in order to be monitored on one or more QDA-recorders (1…6).

Like for all other output resources to QDA the names of the outputs can be altered individually by doubleclick on the output name after it has been placed in the output margin area, e.g. "Value CH #1" → "Tension 1". Once they are connected, they are „piped" to QDA. Note that the variable names are piped too, so that QDA would address the signals by their names which you see within the output resource area.

Remark: If you load another logic plan (layout), the resources will not be updated with the specific variable names included in the plan, but the logic plan itself has the given names. So, QDA will always have the correct assignment to signal (i.e.) but within the resource window it would be called by its default name, e.g. Value CH 1. Because the channel information data are transmitted every minute, it may be helpful to restart QDA in order to shorten the update time.

5.2.7.2. 3X-Channels for QDA and ibaVision3X

There are two 3D-channels "Flatness 1" and "Flatness 2". These channels are dedicated to the QDA 3x-window. So, the data, coming from a flatness control system (e.g. SIEMENS Flatness-PC), only need to be linked with the 3x-channel.

- ❑ - ibaLogic supports up to 2 * 128 3D-channels
- ❑ - QDA provides a 3D-window
- ❑ - ibaVision3x supports an unlimited number of windows to be supplied by the same ibaLogic-pipe.

The control variables for the QDA 3x-display are described in the following section.

5.2.7.3. Variables

The set of variables is used for remote parameterization of QDA. Usually, the variables are part of an input TechnoString which comes by TCP/IP or directly from the PLC. There are three components of QDA which refer to the variables.

- QDA File storing and rolling material information
- QDA 3X-window scaling
- QDA FFT window roll stand symbols

Variable name / resource	Meaning in ibaLogic	Action in QDA, if connected	Remark
counter	number of received telegrams [float]	none	
Data version number	Version of data set [string]	none	
Time stamp	Actual time of data set [string]	Time stamp in recorder strip	
Strip id	"name" of coil [string]	Names file to be stored when selected in the QDA data storage setup menu (create file name by strip id enabled)	With the ability of string operations, ibalogic can combine the incoming strip id with a trigger counter to create "new" filenames
Strip length	Estimated strip length (constant for one strip!) [float] [m]	Scales the x-axis of length based QDA strips and the 3X-window (static option in QDA general properties enabled)	If strip length changes while recording a coil ,QDA can run into performance problems because of continuously new scales of the x-axis. Lowest possible strip length is 200m!
Head length	Length of strip head [float] [m]	none	
Tail length	Length of strip tail [float] [m]	none	
S1: Diameter BUR top : **S5: Diameter BUR top**	Dimensions of the top backup rolls for stands 1 to 5 [float] [mm]	All these geometric dimensions control the behavior of the stand symbols in the QDA FFT-window. With known stand speeds and gear ratios, possible "excentrities" of rolls can be detected.	
S1: Diameter BUR bottom	Dimensions of the bottom backup rolls for stands 1 to 5 [float] [mm]		
S4: Diameter IMR top **S5: Diameter IMR top**	Top intermediate roll diameters of stands 4 and 5 [float] [mm]		
S4: Diameter IMR bottom **S5: Diameter IMR bottom**	Bottom intermediate roll diameters of stands 4 and 5 [float] [mm]		
S1: Diameter WR top **S5: Diameter WR top:**	Dimensions of the top working rolls for stands 1 to 5 [float] [mm]		
S1: Diameter WR bottom : **S5: Diameter WR bottom**	Dimensions of the bottom working rolls for stands 1 to 5 [float] [mm]		
S1: Thickness set point : **S5: Thickness set point**	Estimates thickness after stands 1 to 5 [float] [mm] [mm]	none	
S1:Reduction (out of rolling directive) : **S5: Reduction (out of rolling directive)**	Reduction factor between incoming and outgoing material of a stand in percent [float] [%]	Controls the length forwarding speed of the static strips in QDA.	These factors normally come from the L2 control system
Small zones **Wide zones** **First zone** **Last zone**	Define the number of small and wide zones of the flatness measuring system or similar multi transducer systems [float] [-] First zone/last zone control the displayed width – cut the zones where no material flow is detected	Number of wide zones (in the middle of the strip) and small zones (at the edge of the strip) to control the 3X-window layout. Note the sum small zones + wide zones must be identical to the number of connected 3X-Signals!	
Message Rec. 1 : **Message Rec. 6**	Name for a recorder strip [string]	The message is displayed in the corresponding recorder strip message box	
Variables 90 to 97	Recorder status controls 1 to 8	Connected to either recorder window 1 to 8.. A log. 0 disables the recorder movement, a log. 1 enables it.	In some QDA versions these resources must be wired and enabled to enable time based recorder movement! **UseRecStat = 1**
Variable xyz	Reserved signals	not yet connected to QDA should not be used !	

5.2.7.4. Controls

The control resource set actually supports four different functions which control the QDA recorder.

- *Start Acquisition*: Starts the QDA recorder with transition from FALSE to TRUE
- *Stop Acquisition*: Stops the QDA recorder with transition from FALSE to TRUE
- *Pause Acquisition*: Pauses the recorder while this signal is held TRUE
- *Print*: Prints a hardcopy of the actual screen.
- *Save CAM*: Stores the CAM contents
- *Length Trigger*: Meter pulse for QDA trend-window (length-based statistics)
- *Head*: TRUE to mark the phase when the strip head is rolled
- *Steady state*: TRUE during the phase of "Steady state" (constant operational conditions)
- *Tail*: TRUE to mark the phase when the strip tail is rolled, initiation of corresponding calculations

The pause function gives you the advantage to save recording capacity while the mill is stopped e.g. for repair without losing the relation between the coil in the mill and the corresponding file. While pause is active the recorder just waits until pause is disabled again to continue recording in the same file. Because of this behavior, it is obvious that it would be helpful to detect this pause later in the analyzing phase. To do this, just delay the pause signal for a few milliseconds with the help of ibaLogic and record the not delayed pause signal.

5.2.7.5. Material tracking (QDA Recorder #6 controls)

To control the complex functionality of an online length-based material tracking screen a set of controls has been implemented. The counterpart of this functionality is the recorder #6 in QDA.

The controls are split in 2 sections – Feeds and Triggers:

- ***Feeds (1...8), real:***

These controls monitor the material flow to each of the 8 recorder strips in recorder #6. Note that each feed corresponds exactly to one strip in this recorder!

- ***Triggers, boolean:***

Indicate that the "material" has reached just this position

Example:

If "Feed 1" shall control the flow of the material which leaves stand 1 (shown in recorder #6, strip 1, where counting begins with strip 0) the "Trigger 1" would indicate that the material has just reached this position.

This part of ibaLogic + QDA requires a good knowledge of the process and the process control system to establish this part of functionality.

5.2.7.6. Strip Tags

This resource set is used to control the QDA strip label contents. For every recorder (6) and every strip within a recorder (max 8) a control (data type string) is available. So, ibaLogic is able to control the label written to the QDA online (and offline) display. The labels will be stored by QDA. At every start trigger event or when the strip tag contents has changed the string is transmitted to QDA.

Any ASCII string can be sent to QDA (max 10 characters).

A maximum of 20 labels (prints) per strip and screen is provided by QDA (e.g., if you define a label which changes every second, and the QDA screen is set to monitor 60 seconds, you would see 20 labels moving over the screen from left to right or vice versa).

5.2.8. Reflective Memory (RM)

The link between RM-resources and RM-interface is part of the PCI-configuration as described in chapter 2.6.5.

Each of the 32 RM-output modules consist of 32 output signals whose signal names are clearly assigned to the modules. Additionally, each signal has a description (text) which can be edited in order to improve the technical comprehension by the user.

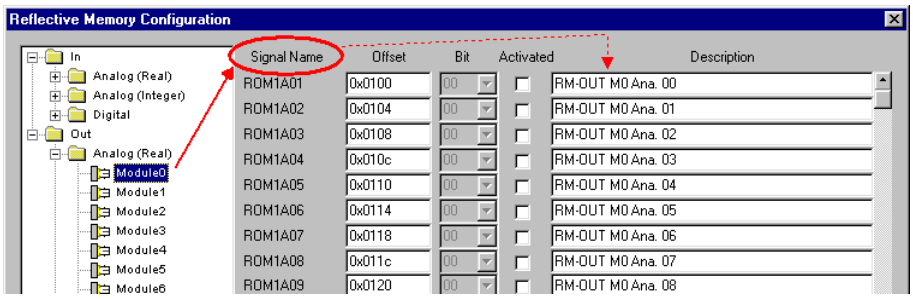


Fig. 81 Reflective Memory output resources, connection between module, signal name and description

The descriptions of the output signals appear also in the resource tree and further in the layout when the signals are used. They also can be found in the tooltip when placing the mouse cursor over a corresponding connector.

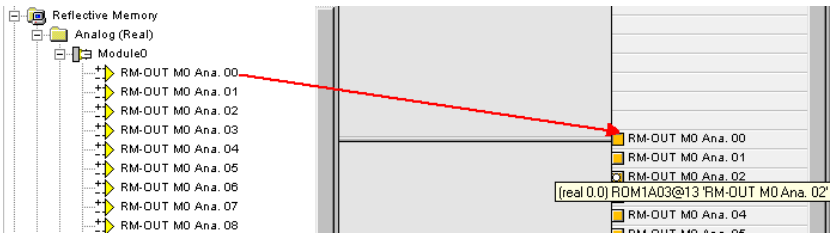


Fig. 82 Reflective Memory output resources, appearances of signal description

5.2.9. eCon/PPIO OUT – outputs to eCon / eCon32

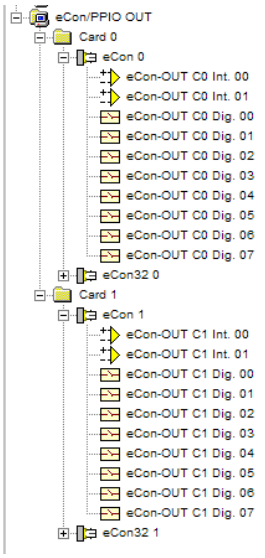
These output resources are dedicated to the eCon and eCon32 devices from iba.

The eCon devices are small I/O devices which have to be connected to a PC via the parallel printer port. There are two types available:

eCon: This type consists of 2 analog outputs (AO) and 8 digital outputs (DO).

eCon32: This device provides 32 digital outputs.

Up to two of these devices can be operated in combination by one parallel PC port.



The assignment of eCon devices and output resources is as follows:

Card 0	first eCon at parall port
	if eCon, then 2AO and 8 DO if eCon32, then 32 DO
Card 1	second eCon, connected in line to the first eCon
	if eCon, then 2 AOI and 8 DO if eCon32, then 32 DO

The digital/analog conversion of the analog output values is based on a 10-bit resolution (step width on digital side = 64)

Because the system is not able to detect the type eCon device which is connected, the correct settings have to be made in the system settings (↪File ↪System settings ↪Parallel).

↗ *See also chapter 2.5.3*

Connected with the selection of the eCon device is a so-called zero mask. The zero mask forces all outputs of the eCon to zero (0) when the ibaLogic layout is stopped or switched to offline mode. (safety reasons)

Concerning the analog outputs please note that the output value 0 (zero) corresponds to a hexadecimal value of 0x8000 (high + lowbyte) in the zero mask. The eCon devices have an analog output range from –10 V to +10 V. A zero mask of 0x0000 would cause an output value of –10 V.

For a better comprehension of the connections between hex-code and output assignment please refer to Fig. 83 on the following page.

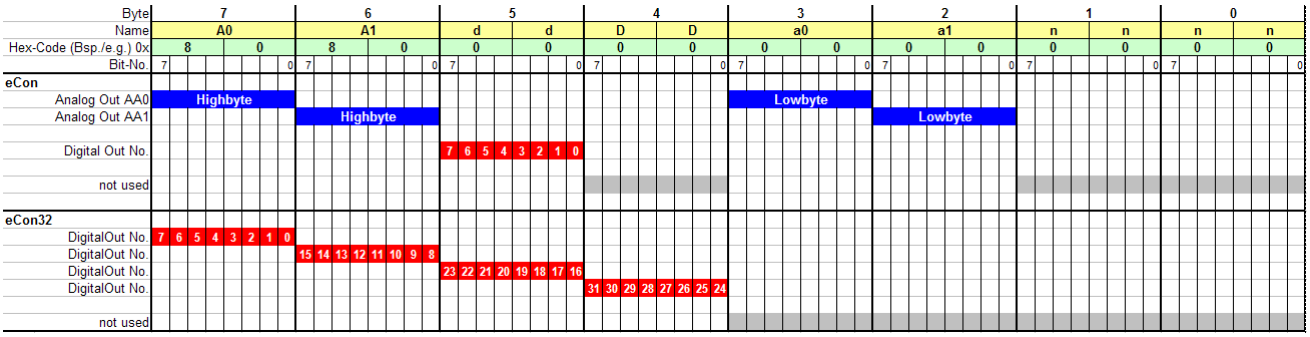


Fig. 83 Hex-addressing of analog and digital outputs for eCon and eCon32 (zero mask)

For further informationen concerning the eCon devices please refer to the related hardware documentation. That documentation also cares about the software engineering.

hw_man_econ_en_A4.pdf

5

5.2.10. Playback OUT

One difital output is provided for the playback mode of operation but it's only for internal use. This output may be set on TRUE or FALSE by the ibaLogic layout in order to control the playback of a dat-file, respectively to restart the playback.

Playback Out Restart, if set on TRUE (impulse), the "cursor" for replaying the data file is reset to the file beginning (first sample) and the playback starts again.

5.3 OPC - Communication

The intention of the OPC standard interface (**O**LE for **P**rocess **C**ontrol) is to advance the integrated use of automation and control systems, field devices and office applications.

Meanwhile, the OPC interface, which was specified by the "OPC foundation", is considered as a powerful interface in the Windows® environment and it is supported by many users and manufacturers. OPC is based on the OLE/COM technology from Microsoft Corporation. According to the OPC specification there are two interface definitions: the "Custom Interface" and the "Automation Interface".

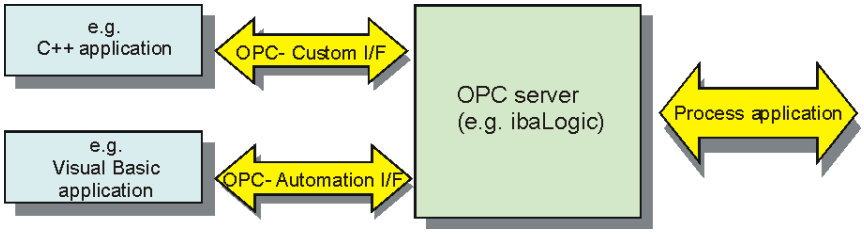


Fig. 84 OPC-Interfaces

As an OPC client, e.g. a Visual Basic-application communicates with the OPC server by the "Automation Interface". (see also part B, "References": [4], [5])

In order to explain the process of OPC communication the interaction between ibaLogic and a Visual Basic application is taken for example in the following.

5.3.1. OPC Automation Server Object Model

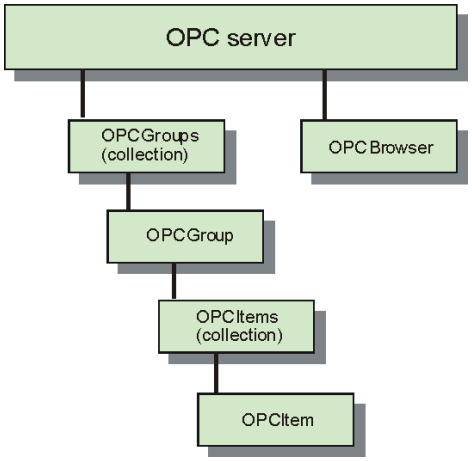


Fig. 85 OPC Automation Server Object Model

Object	Description
OPCServer	A client has to create an instance of the OPCServer object first. Then the client must connect this instance with the OPC Data Automation Interface (method 'connect'). Now, the OPCServer object can be used to get general information from the server and to create and manage OPCGroup objects.
OPCGroups	This is a collection of all OPCGroup objects which were created by a client within one OPCServer object including their methods of creation, cancellation and management. It also contains the default properties of the OPCGroup objects at the time of their creation.
OPCGroup	An OPCGroup object is a mean to organize data, e.g. an operator screen or a report. The client requests only the data which are related to the screen, resp. report, using a specified transmission rate.
OPCItems	This is a collection of all OPCItem objects which were created by a client within an OPCServer object including their methods of creation, cancellation and management. It also contains the default properties of the OPCItem objects at the time of their creation.
OPCItem	An OPCItem represents a connection to a data source in the server. Every item consists of a value (type Variant), state information and timestamp.
OPCBrowser	An OPCBrowser object shows the hierarchy which has been installed on th eserver, i.e. the branches and items. The browser function is to be used optionally.

5.3.2. Installation of the OPC Driver-DLLs

Before starting a communication between ibaLogic and a Visual Basic application it is required that all participating PC workstations have the same OPC DLLs (DLL = Dynamic Link Library) installed. The following DLLs are required:

Opcproxy.dll	size: 76kB	date:	11/27/02
Opccomn_ps.dll	size: 60kB	date:	11/27/02
Opcdaauto.dll	size: 156kB	date:	11/13/00

These three files can be found on the ibaLogic CD-ROM in the folder \sample_OPC_VB_V103\OPC_Install\OPC_DLL's.

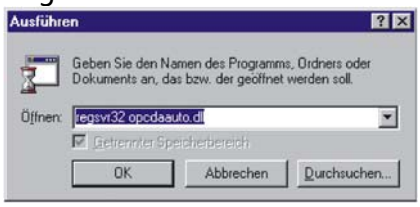
In order to register the DLLs on your PC please follow these steps:

1. If you have an up-to-date version of the ibaLogic CD-ROM you'll find there a DLL-installer program (install.exe) in the folder *sample_OPC_VB_V103\OPC_Install*. Just execute this program.
2. ...or copy the entire folder *sample_OPC_VB_V103* from CD in a folder of your choice on the harddisk of the ibaLogic-PC.
3. Browse in Windows Explorer for the program...\ *sample_OPC_VB_V103 \OPC_Install\Install.exe* and start it by a doubleclick. The successful registration of the DLLs will be posted.

If you don't have access to this install program (e.g. with older installations of ibaLogic) then proceed as follows:

1 Copy the above mentioned DLL-files into the system-folder on the harddisk of all involved PCs: "c:\Winnt\System32" (Windows NT) or c:\windows\system32 (Windows XP) respectively.

2 Afterwords, the DLLs have to be registered one by one, using the command "regsrv32". Use the Start-button in the Windows task bar and *Execute...*



3 Make sure, that the three DLL-files are installed on the OPC server (ibaLogic) and on the OPC client (Visual Basic), too. In case of using a single workstation, the DLLs need to be installed only once.

5.3.3. OPC-sample application with Visual Basic

OPC-VB sample application (sample_OPC_VB_V103)

For a better understanding of the OPC-related functions in ibaLogic you'll find a simple sample application on the ibaLogic CD-ROM in the folder \sample_OPC_VB_V103.

This folder contains all necessary programs and files for running the ibaLogic application. An installation of Visual Basic (VB) on the PC is not required.

For those of you who'd like to examine the Visual Basic application (project) and like to reuse parts of it for their own projects, the relevant programs and files are stored on the CD as well. In order to open the VB-project an installation of Visual Basic (Visual Studio) on the PC is required.

Please follow these steps to run the sample application:

1. If not done yet, please copy the entire folder *sample_OPC_VB_V103* from CD into a folder of your choice on the harddisk of the ibaLogic-PC.
2. Start ibaLogic.
3. Open the sample application from the folder ... *\sample_OPC_VB_V103\LYT-File**sample_layout_OPC_VB_V103.lyt***
4. Switch the layout online by clicking on  (pink background color).
5. In the Windows Explorer start the VB-project *\sample_OPC_VB_V103\VB_application\sample_application_OPC_VB_V103.exe* with a doubleclick.

A new window should appear on the screen, showing the values of the OffTask-connectors in the ibaLogic layout.

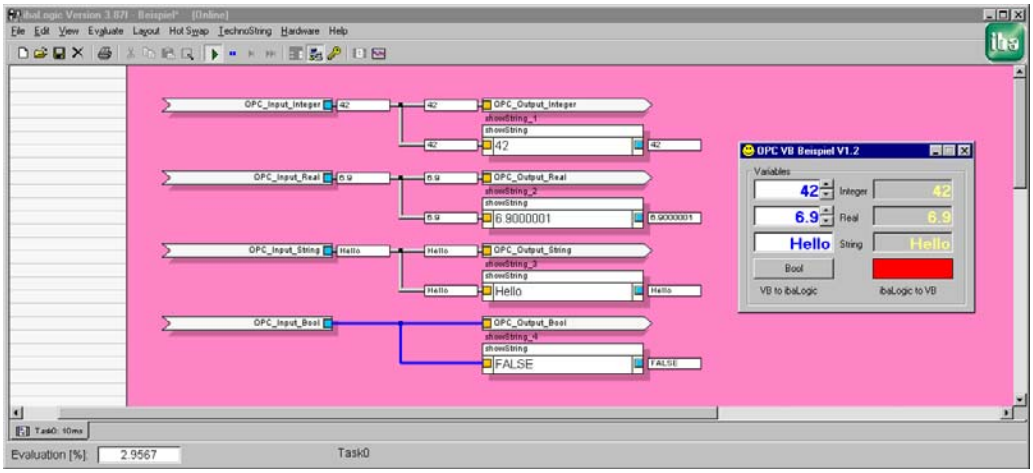


Fig. 86 OPC sample application, windows

The following OffTask-connectors are defined as outputs to Visual Basic:

OPC_Output_Integer	Output INT as displayed value
OPC_Output_Real	Output Real as displayed value
OPC_Output_String	Output of an ASCII character field (text)
OPC_Output_Bool	Output as boolean variable (here red/green)

The following variables may be entered in the VB operator- and display window and called up in ibaLogic:

OPC_Input_Integer	Field for entry and display of an integer value in ibaLogic (enter values by using the up/down arrow buttons)
OPC_Input_Real	Field for entry and display of an real value in ibaLogic (enter values by using the up/down arrow buttons)
OPC_Input_String	Field for entry and display of a text (ASCII-string)
OPC_Input_Bool	Button for switching (toggle) in ibaLogic

The next figure illustrates the connections of data flow between ibaLogic and VB and shows the settings of the OffTask-connectors.

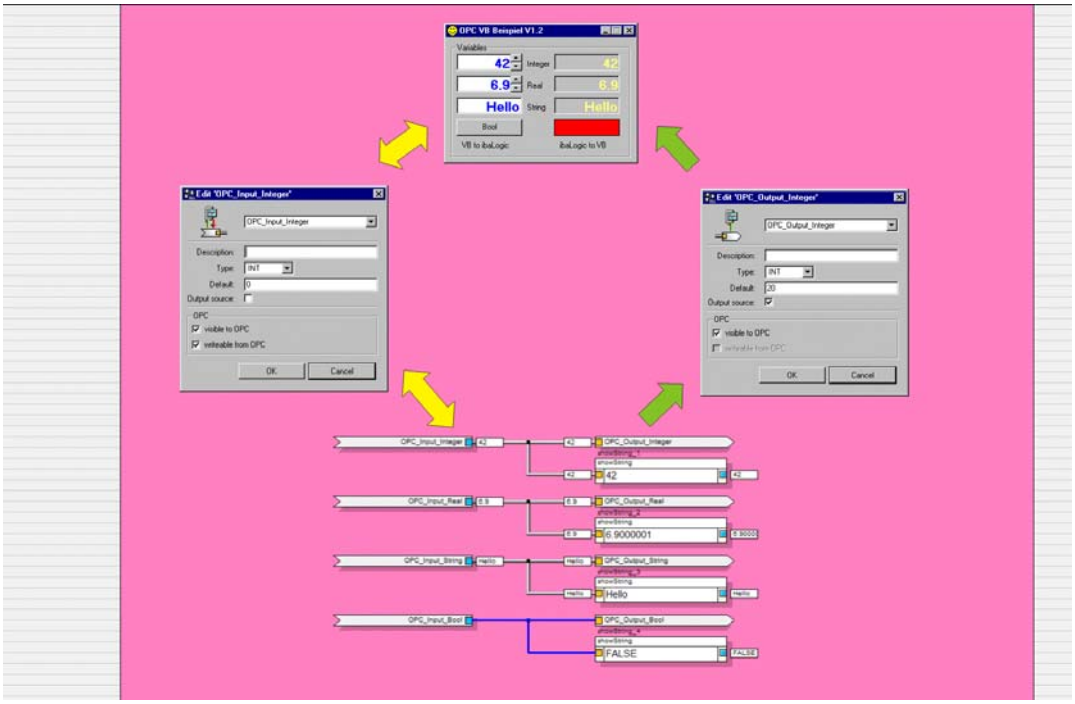
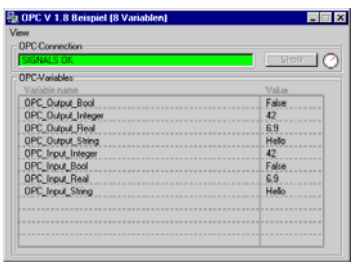
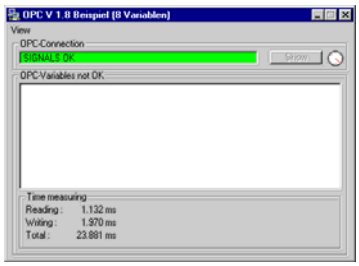
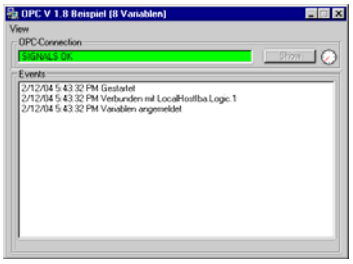


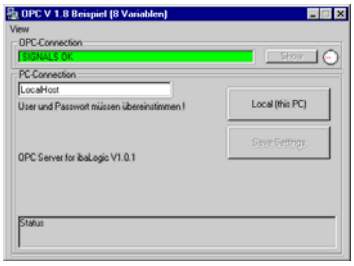
Fig. 87 OffTask-connector settings for communication between ibaLogic and Visual Basic

OPC-Diagnostics









The OPC-connection monitor is a helpful tool for checking the OPC-VB-communication. This tool is not part of iba-Logic but of the VB-sample project on the CD.

In four different views (selection over menu) information is available about:

- quantity, names and values of OPC-variables
- status ok / error
- duration of read- and write access cycles
- event history
- PC-connection

5

If OPC-client (ibaLogic) and OPC-master (e.g. a HMI-system) are running on different PCs, communicating over network, please note the security settings when working under Windows XP.

See also chapter 6.2.3.

6 Installation

6.1 Installation of ibaLogic

6.1.1. Installation with install wizard (for eCon only)

1. Insert the ibaLogic insatllation disk into the CD-ROM drive of your PC. The installation wizard starts automatically. If not, please execute the program *Setup.exe* on the CD.
2. Choose your preferred language. The language selection will not only affect the installation dialog but also the documentation and sample applications which are copied to your harddisk.
3. Follow the messages of the installation program.
4. Eventually, a new dialog opens in order to select a parallel printer port which may be connected to an eCon device. Select the port from a field (left) in the dialog. Under Windows XP you can check the availability of parallel ports in your system by using the device manager.
   If you need more information about the setup of the parallel port just click on the corresponding button. Close the dialog with *Next*.
5. In the next step select the type of eCon device which may be used. If you plan to use only one eCon, just select the first one (left). When using two eCons select both. Click on *Next*.
   You can change these settings any time later.
6. Click on *Finish*.

6.1.2. Standardinstallation from CD

1. Create a folder on the harddisk of your PC, e.g. c:\ibaLogic.
2. Copy the entire folder *ibaLogic*\ from CD into that folder on your harddisk.
3. If you are working under Windows NT please remove the read-only attribute from the files after copying. This not necessary when you are using Windows XP.
4. If you have received an ibaLogic update by email or if you have downloaded a new release from the web, please extract all files from the zip-file into the ibaLogic program folder on your harddisk.
5. Start ibaLogic with a doubleclick on *..\ibaLogic\ibaLogic*Version*.exe* in the Windows Explorer or use the execute command in the start-menu of Windows. ibaLogic will create all required subdirectories.

The folder *ibaLogic\configuration\schematics* is the standard folder for ibaLogic application programs which are to be stored as layout (*.lyt) and Structured Text (*.txt).

The folder *DLLs* will later contain all DLLs and the folder *FBs_Macros* will contain all function blocks and macroblocks that will be created during engineering.

Please feel free to establish a shortcut for ibaLogic on the desktop or in the program-start-menu.

This can be useful if the PC is used rather for engineering than for real online process control, because other programs could be also used on it. But on a PC which is dedicated to the control of processes or machines, no other PC-application should be installed (such as office tools, games etc.). In that case an ibaLogic call in the Windows autostart folder is enough and recommended.

6.2 USB dongle

Due to the wide and increased availability of USB-interfaces in PC-systems iba offers also the software hardlock (dongle) for USB-sockets. As an advantage the serial interface can be used for other applications, e.g. for control connections to an UPS (**U**ninterruptable **P**ower **S**upply), or for communication.

USB is generally supported by Windows XP. USB is usually not supported by Windows NT, Therefore a manual installation is required.

6.2.1. USB dongle and Windows XP

The support for USB-Dongles is to be installed autoamatically by the iba software products, such as ibaPDA, ibaLogic, dongleupgrade etc.

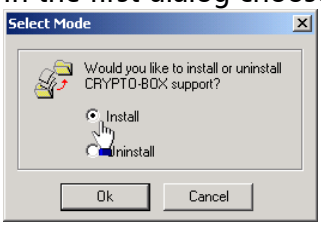
6

A manual installation is not required.

6.2.2. USB dongle and Windows NT

In order to install manually the USB support on an existing Windows NT installation, please follow these steps:

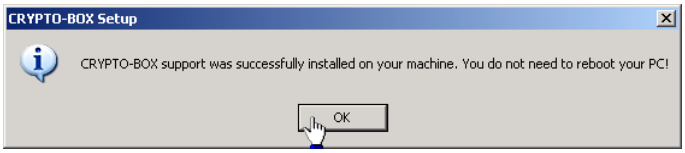
1. Start the program CBSETUP.exe which has come with the dongle and ibaLogic on the CD-ROM.
2. In the first dialog choose *Install* and click *Ok*.



3. Choose *Yes*

4. Select *CRYPTO-BOX USB* and click *Ok*.

5 Depending on the operating system you'll be informed if a reboot of the PC is required for the installation to come into effect. Usually, a reboot is required with Windows NT, with Windows XP it's not.
Confirm the last message with *Ok*.

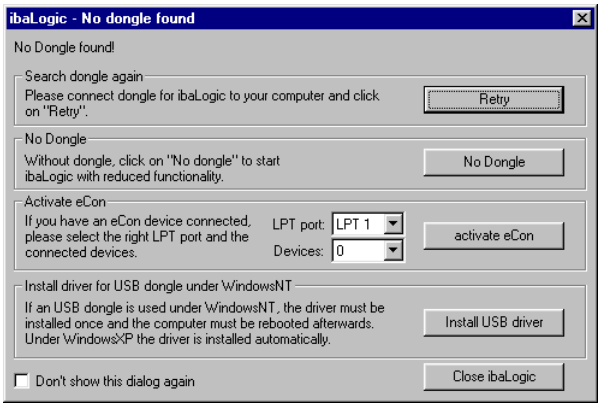


There is also batch routine to get the USB support installed. If you like to use this routine start the installation as follows:

CbSetup.exe /q /CRYPTOKEN

Please make sure that the USB-interface is enabled in the BIOS of your PC.

If you start ibaLogic and see the following dialog, the reason may be a plugged USB dongle which could not be detected by ibaLogic because of a missing USB support.



In that case, click on the button *Install USB driver.*

6.2.3. Security settings in Windows XP

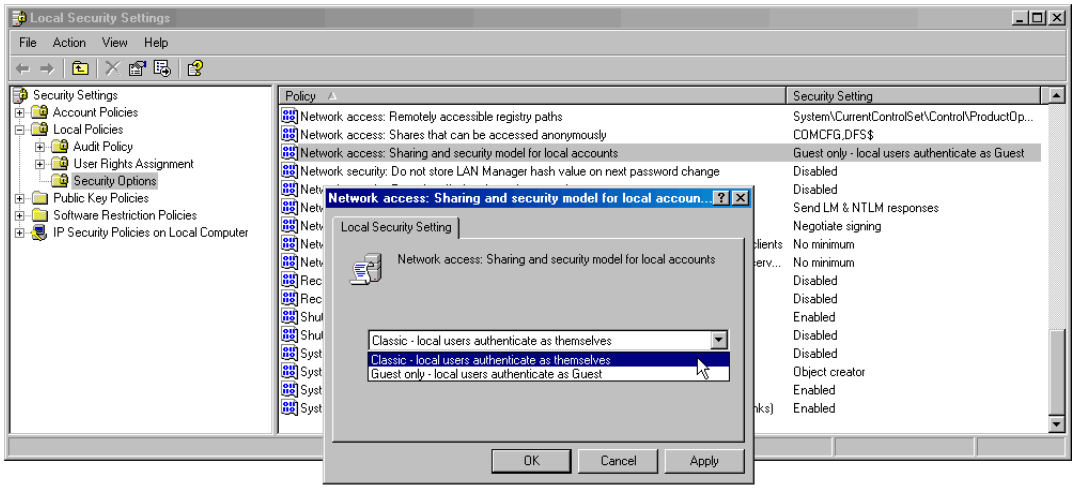
Some settings concerning communication and networking are more restrictive in Windows XP compared to Windows NT or other former Windows releases.

Some features of iba's software products take advantage of distributed PCs connected by a network, for example:

- ibaLogic connections between OPC-client and OPC-server
- Start of ibaAnalyzer triggered by a remote PC, running the postprocessing command by a DatFileWrite-function block in ibaLogic
- Use of postprocessing command in ibaPDA to remote PCs
- Remote diagnostics with ibaDiag

Some settings should be considered in order to guarantee the proper function of these services when all or some workstations run on Windows XP in a network:

1. If possible, the Windows login username, password and user rights should be the same on all involved workstations, sharing these services.
2. If there are different logins used on the different workstations, the user must be registered in the user administration of the participating workstations vice versa, including name, password and rights.
3. On every workstation the parameter *Network access: Sharing and security model for local accounts* (in the Windows local security settings) should be set on ***Classic***.
   You'll find this parameter under Windows XP Start menu ↪*Settings* ↪*Administrative Tools* ↪*Local Security Policy* ↪*Security Settings* ↪*Local Policies* ↪*Security Options.*



6.3 System configuration for ISA-cards

For setup of the ISA-driver configuration in ibaLogic select menu ↪File ↪ISA Configuration

(This command is disabled in Windows XP when no ISA card had been detected.)

The following dialog opens:

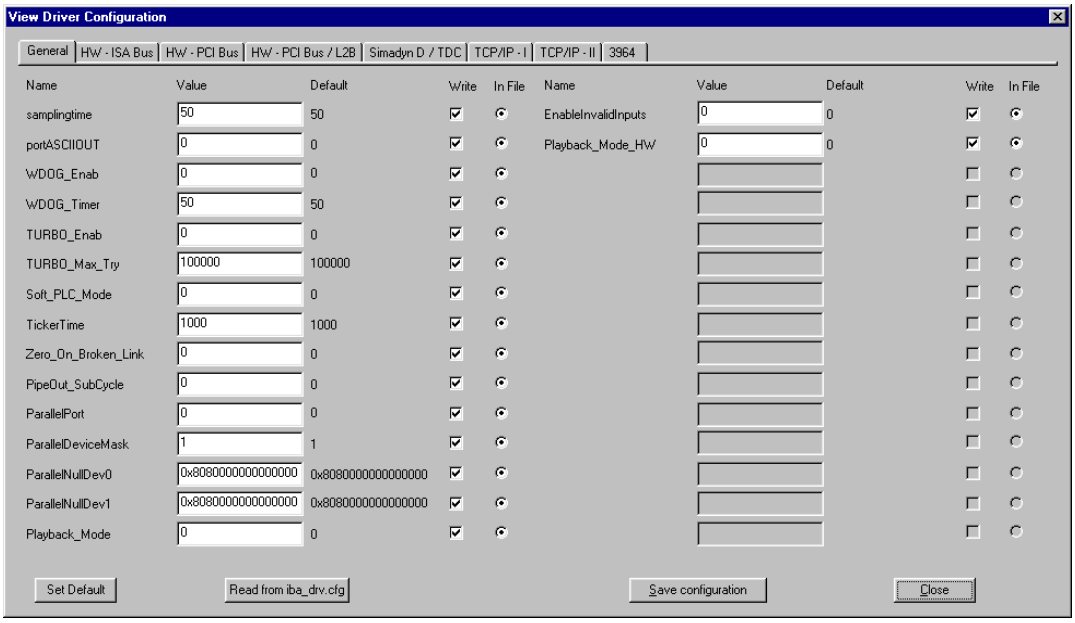


Fig. 88 ISA-card configuration

Here, the hardware parameters are to be entered.

If you are using ISA-cards go to the tab "HW – ISA Bus" and enter the parameters according to the hardware settings on the card(s).

Furthermore, the basic sample time has to be entered under the tab "General" (default setting: 50 ms).

If you use, for example, a FOB-F card the settings must be done as follows:

- portFOBF = 1
- FOBF_AcqAddress = D8000
- Int_Vector = 5

Note: The settings are only valid in compliance with the hardware settings on the card (bridges). For the first start of ibaLogic only the entries under the first tab are required.

In order to save the settings, press the button "Save configuration". ibaLogic will save the settings in the file iba_drv.cfg.

Restart ibaLogic to apply the changes.

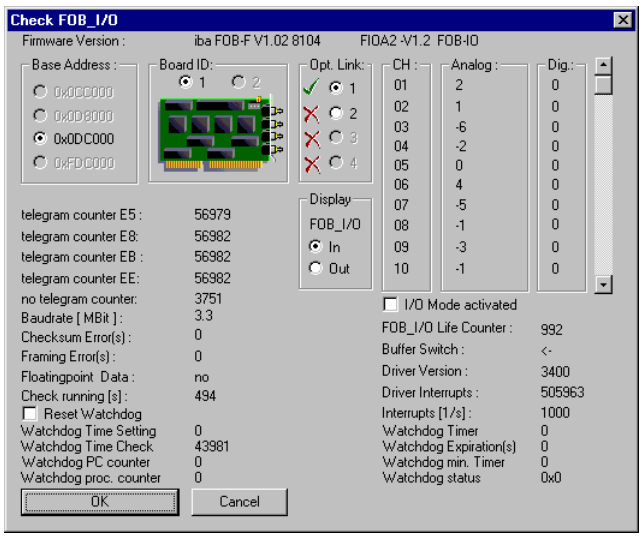


Fig. 89 ISA-card, check hardware settings

Check the hardware settings with the menu ↪*Hardware*.

The card is working properly if the interrupt counter shows a rate of approximately 999 – 1001 interrupts per second.

If a connection has been established between the card and a connected Padu, the red crosses will be replaced by a green checkmark √ at *Opt. Link*.

6.3.1. Recommended ISA hardware settings

Switch off your computer and unplug it from the power supply. Then open the chassis of your computer. Therefore refer to the requirements written in your PC manual.

The hardware components may be permanently damaged by electrostatic discharge. Use the required safety precautions to handle hardware components.

There are several possible hardware configurations. A maximum of three cards is supported. It is necessary to install the hardware as shown in the following table. Note which card has to be selected as interrupt master (underlined and marked red). Note that only 2 ID´s are supported for one card address! Note further that the PCMCIA card is not supported! FOB-F can be replaced by FOB card.

#	Application	Card 1	Card 2	Card 3
1	Simadyn access Two SD connections	FOB SD * CS22 address is 0xE0000 mostly ID must be 0	-	-
2	Simadyn access 3 or 4 SD connections **	FOB SD * CS22 address is 0xE0000 mostly ID must be 0	FOB-SD 0xE0000; ID = 1 Do not forget the cascade connector!	FOB-SD Not supported
3	Simadyn plus (several) FOB´s	FOB SD * CS22 address is 0xE0000 mostly ID must be 0	FOB-F 0xDC000 ID = 0 No IR	FOB-F 0xDC000 ID = 1 No IR
4	Simadyn plus Profibus	FOB SD * CS22 address is 0xE0000 mostly ID must be 0	L2B-F Address: D8000 ID = 0 No IR	
5	Simadyn plus Profibus plus FOB	FOB SD * CS22 address is 0xE0000 mostly ID must be 0	L2B-F Address: D8000 ID = 0 No IR	FOB-F Address: DC000 ID = 0 or 1 No IR
6	Flatness PC or Profibus application	L2B-F *** Address: 0xD8000 ID = 0 Internal interrupt	-	-
7	Flatness PC or Profibus application	L2B-F *** Address: 0xD8000 ID = 0 Internal interrupt	L2B-F *** Address: 0xD8000 ID = 1 No IR	-
8	Flatness (or Profibus) plus (several) FOB´s	L2B-F *** Address: D80000 ID = 0 No IR	FOB-F Address: DC000 ID = 0 External or Internal IR	FOB-F Address: DC000 ID = 1 No IR
9	Several FOB´s	FOB-F Address: 0xDC000 ID = 0 External or internal IR	FOB-F Address: 0xDC000 ID = 1 No IR	FOB-F Not supported
10	Notebook applications	PCMCIA-F****	IR always-	-

A 2/2 IO FOB can be handled like a FOB F. Note however, that the interrupt has to be set to INTERNAL only!

* To ensure proper function (i.e. FOB SD) make sure that Segment E is not used, because the FOB-SD card will use the whole segment!

** More than one connection means that ibaLogic accesses different CS1x cards. It is not possible to hook up more than one connection to a CS1x!

*** Check Profibus DP Slave address properly corresponding to programmed application (i.e. S7) and make sure the mode selection (S7 integer, flatness ....) is made correct (see also L2B manual). Only two flatness channels are supported by ibaLogic and QDA! When configured as S7 DP Slave, the L2B acts like a FOB-F and must be treated correspondingly.

**** For software installation of PCMCIA-F see also PCMCIA_F manual.
Never select address range CC000 because if the PC motherboard supports onboard SCSI these addresses might be in use!

6.3.2. The Configuration File "iba_drv.cfg"

Before mounting the cards check the address entries and configure your hardware with the addresses you find at your *iba_drv.cfg* printout:

```
....
portFOB = 0                     // every "1" indicates that such a card is present in
portFOBF = 0                    // the PC (here: FOB-IO)
portFOBSD = 0
portPROFI = 0
portFOBIO = 1
portASCIIOUT = 0
PCMCIA = 0
....
FOB_AcqAddress = 0xDC000        // for FOB cards
FOB_AcqLength = 0x440
FOBF_AcqAddress = 0xDC000       // for FOB-F cards and FOB IO cards also !!!!
FOBF_AcqLength = 0x3100
FOBSD_AcqAdress = 0xE0000       // the FOBSD needs 64kbytes of memory !! check it!
PROFI_AcqAddress = 0xDc000             // for FOB L2B cards
PROFI_AcqLength = 0x440
PCMCIA = 0                      // for PCMCI-F cards
CS22_BgtName = PDA001           // following parameters for FOB-SD and CS22 only !
CS22_AcqAddress = 0xD0000
Simadyn_Sync_Timeout = 15
Simadyn_Proc_Timeout = 15
CS22_0_OwnName = DPDA1A                // all these parameters must be set for CS22 and
CS22_0_Partner = D0900B         // FOB-SD accordingly
CS22_0_SoftwareVersion = V420
CS22_1_OwnName = DPDA2A
CS22_1_Partner = D0900B
CS22_1_SoftwareVersion = V420
CS22_2_OwnName = DPDA3A
CS22_2_Partner = D1200B
CS22_2_SoftwareVersion = V430
CS22_3_OwnName = DPDA4A
CS22_3_Partner = D1500B
CS22_3_SoftwareVersion = V430
CS22_Nboards = 0                // here only (!)the number of CS22 must be set
```

Note: Two FOB cards can have the **same address** but must then be „named" with two **different board ID´s** (0 and 1 are possible and are supported by the driver actually). At max 2 FOB cards can be installed within one PC. There must be always one card with the **ID 0** driving the process interrupt Therefore select either Interrupt from connected PADU´s – in this case the interrupt-switch must be turned in direction „outside" of the PC, or use the card internal interrupt source – then the switch must be in position directing „inside" the PC. The first option has the advantage that the optical link and the PADU is monitored. Any broken link is immediately detected. In the case of a broken link the FOB will automatically generate a „default" interrupt but at a much lower frequency. If ibaLogic would act strange online (very slow) the check if the connectivity is o.k. or if the PADU is switched ON.

Note: The FOB 2/2 IO must be always configured with **Interrupt internal** to work properly.

If you like to choose different addresses do not forget to modify the iba_drv.cfg file correspondingly!

FOB-SD is an iba card. Note that the FOB SD always needs a free space of 64kbytes in your PC. Note further, that Windows diagnostics does not always have the correct status of free memory. It can happen although the requested memory block is marked as free by Windows while the block is not entirely free.

This would seriously affect the FOB SD operation. FOB-SD also must generate the process interrupt.

Do not forget to check all the Simadyn parameters !

Fix all the necessary screws, close the PC rack, boot the PC and start ibaLogic.

6.3.3. System Configuration with PCI-Cards

The hardware installation of the cards is described in the documentation which comes with the cards.

After the cards have been installed correctly in terms of PC Slots and PCI-interrupt, start ibaLogic and check the system settings.

Under menu ↪*File* ↪*System settings* press first the button "Autoconfig" in order to obtain the basic settings for the system.

Then check the different tabs and make sure that unused cards are disabled and then configure the cards which are used. The dialog windows for the card configuration can be opend by pressing the button *Configuration*... in the lower right corner or by using the menu ↪*File* ↪*PCI Configuration* ↪*card*. (see also chapter 2.5)

In case of using a FOB-SD / -TDC card, this card should be configured as interrupt master.

7 Additional information and examples

7.1 Sample listing for DLL creation

Please note also the remarks in chapter 3.12.

Due to print-related technical reasons some lines in the following listing are wrapped. Please read carefully.

7.1.1. dllForm.hpp

```
//************************************************************
//
// Filename:  dllForm.hpp
//
// Author:    Dipl.-Ing. Hubert Andris
//
// Created:   05-Sep-1998
//
// Description:
//   Interface definition for DLL Forms.
//
// External Definitions:
//   DLLExport      compile for DLL export rather than for DLL
import
//
//************************************************************

#if !defined(DLLFORM_HPP)
#define DLLFORM_HPP

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include <windows.h>
#include <assert.h>

#if defined(DLLExport)
#define DLL __declspec(dllexport)
#else
#define DLL __declspec(dllimport)
#endif

#define DLL_INTERFACE_VERSION_HIGH  (1 << 16)

////////////////////////////////////////////////////////////////
// Define function prototypes for DLL functions used.
//
// Naming convention:
//   PF<type><arg1><arg2>...
//
//   where
//     type  = V  returns void
//             I  returns int
//             W  returns DWORD
//             S  returns short
//
//     arg#n = I  arg#n is int
//           = PI arg#n is pointer to unsigned int
//           = PC arg#n is char pointer
//           = PV arg#n is void pointer
//
typedef void  (*PFV)(void);
typedef int   (*PFI)(void);
typedef DWORD (*PFWV)(void);
typedef int   (*PFII)(int io);
typedef short (*PFSII)(int io, int index);
typedef void  (*PFVPV)(void *ptrData);
typedef void  (*PFVPCI)(char *pName, int cbName);
typedef void  (*PFVIIPCI)(int io, int index, char *pName, int
cbName);
typedef void  (*PFVIIPVIPV)(int io, int index, void *ptrData, int
size,
                            void *pInstanceData);
typedef void  (*PFVIPVIIPV)(int index, void *ptrData, int size,
int valid,
                            void *pInstanceData);
typedef int   (*PFIIPVIPV)(int index, void *ptrData, int size,
                           void *pInstanceData);
typedef void  (*PFVIPVPI)(int cbSize, const void *pGlobal,
                          const __int64 *pMilliSeconds);
typedef void  (*PFVPVIPV)(const void *pGlobal, int cbSize,
                          void *pInstanceData);
typedef DWORD (*PFWPVIPV)(const void *pGlobal, int cbSize,
                          void *pInstanceData);

#define MAX_SERIAL_NO_LENGTH  12

typedef struct
{
  __int64 g_EvalTime;
  __int64 g_EvalDeltaTime;
  int     g_Online;
  int     g_Unlocked;
  DWORD   g_System1;
  char    g_DongleSerialNum[MAX_SERIAL_NO_LENGTH];
} globalVarType;

#if 0
////////////////////////////////////////////////////////////////
// DllMain
//
// Called from operating system on load and unload.
//
extern DLL BOOL WINAPI DllMain(HINSTANCE hInstDLL,    // handle
to DLL module
                               DWORD     fdwReason,   // reason
for calling function
                               LPVOID    lpReserved); // reserved
#endif

////////////////////////////////////////////////////////////////
// GetDllVersion
//
// Each Dll shall have a unique version number
//
// Returns:
//   hiword  major version number: describes the program
// interface
//   loword  minor version number: describes the semantic
//
extern DLL DWORD GetDllVersion(void);

////////////////////////////////////////////////////////////////
// General parameters for subsequent functions:
//---------------------------------------------
//
// Each instance of the Dll form has its own data pointer, which
// points to
// dynamicaly allocated memory of size returned by the previous
// call to
// GetInstanceDynamicDataSize().
//
// Parameter:
//   void *pInstanceData
//
//-------------------------------------------------------------
//
// The Dll has read-only access to global variables of Signal
// Manager
// application, which may be required for the evaluation.
//
// Currently defined global variables:
//
//+--------+-------+---------+---------------------------------+
//| offset | bytes | type    | description                     |
//|--------+-------+---------+---------------------------------+
//|    0   |   8   | __int64 | time in 0.1 milliseconds        |
//|        |       |         | relative to last                |
//|        |       |         | InitEvaluation call             |
//+--------+-------+---------+---------------------------------+
//|    8   |   8   | __int64 | time in 0.1 milliseconds        |
//|        |       |         | relative to last                |
//|        |       |         | call to Evaluate                |
//+--------+-------+---------+---------------------------------+
//|   16   |   1   |  char   | = 0: layer is offline/          |
//|        |       |         | = 1: layer is online            |
//+--------+-------+---------+---------------------------------+
//|   17   |   3   |  none   | unused (all bytes 0)            |
//+--------+-------+---------+---------------------------------+
//|   20   |   1   |  char   | = 0: layer is locked/           |
//|        |       |         | = 1: layer is unlocked          |
//|        |       |         | If locked, DLL shall NOT change |
//|        |       |         | any default value!              |
//+--------+-------+---------+---------------------------------+
//|   21   |   3   |  none   | unused (all bytes 0)            |
//+--------+-------+---------+---------------------------------+
//|   24   |   4   |  DWORD  | reserved                        |
//+--------+-------+---------+---------------------------------+
//|   28   |  12   |  char   | index 0 .. 7  : serial number   |
//|        |       |         | index 8       : = 0             |
//|        |       |         | index 9 .. 11 : reserved        |
//+--------+-------+---------+---------------------------------+
//
// Parameter:
//   pGlobal        pointer to global variables array
//   cbSize         number of available global variables
//
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
// GetInstanceDynamicDataSize
//
```

```
// Each instance of the Dll formula has its own pointer, which
// may point to
// any data of any size. This function returns the size for that
// memory.
// Memory allocation/deallocation is completely performed by the
// calling
// application.
//
// Returns:
//   required memory size in bytes
//
extern DLL int GetInstanceDynamicDataSize(void);


////////////////////////////////////////////////////////////////
// GetDllDescription
//
// Parameter:
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null byte)
//
extern DLL void GetDllDescription(char *pDesc, int cbDesc);

////////////////////////////////////////////////////////////////
// GetCount
//
// Parameter:
//   io  0 = input, 1 = output
//
// Returns:
//   number of inputs/outputs (1..128)
//
extern DLL int GetCount(int io);


////////////////////////////////////////////////////////////////
// GetName
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pName    buffer to receive name as ASCIIZ string
//   cbName   size of buffer (including terminating Null byte)
//
// Note: Each input/output name must to be unique!
//
extern DLL void GetName(int io, int index, char *pName, int
cbName);


////////////////////////////////////////////////////////////////
// GetDescription
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null byte)
//
extern DLL void GetDescription(int io, int index, char *pDesc,
int cbDesc);

////////////////////////////////////////////////////////////////
// GetType
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//
// Returns:
//+-------+--------------+-----------------------------------+
//| type  | iec1131 type | size of value in bytes
|
//+-------+--------------+-----------------------------------+
//|   1   |    BOOL      |      1                            |
//|   3   |    INT       |      2                            |
//|   4   |    DINT      |      4                            |
//|   8   |    UDINT     |      4                            |
//|  10   |    REAL      |      4                            |
//|  11   |    LREAL     |      8                            |
//|  12   |    TIME      |      8                            |
//|  16   |    STRING    |   1024 (including terminating null) |
//|  19   |    DWORD     |      4                            |
//|  22   |    ARRAY     | total element count * size        |
//|       |              | of element                        |
//+-------+--------------+-----------------------------------+
//
// Note:
//   Enumeration valType in 'value.hpp' can be used for
//   convenience.
//   If 22 (ARRAY) is returned, then a call to GetArrayHeader
//   will follow.
//
extern DLL WORD GetType(int io, int index);

////////////////////////////////////////////////////////////////
// GetArrayHeader
//
//   Called if GetType() returned 22 (array type)
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to array header:
//+--------+------+-----------------------------------------+
//| Offset | Type | Description                             |
//+--------+------+-----------------------------------------+
//|  0     | WORD | Element type (see description of GetType)  |
//|  2     | BYTE | reserved (e.g. for synchronization flag)   |
//|  3     | BYTE | reserved (e.g. for valid flag)             |
//|  4     | DWORD| array dimension (1..4)                     |
//|  8     | DWORD| Start Index of 0-th subscript              |
//| 12     | DWORD| Stop  Index of 0-th subscript              |
//| ...    |      |                                            |
//|  8+n*8 | DWORD| Start Index of n-th subscript              |
//|        |      | (n < dimension)                            |
//| 12+n*8 | DWORD| Stop  Index of n-th subscript              |
//|        |      | (n < dimension)                            |
//| ...    |      |                                            |
//+--------+------+-----------------------------------------+
//
//   cbBuf    sizeof buffer in bytes sufficient for max array
//   dimension of 4
//
// Note: These functions are only called once for initial
//       initialization of
//       the dll form instance.
//       The total element count is calculated via:
//       Sum over all subscripts: (n-th stop index - n-th start
//       index + 1).
//       For multi-dimensional arrays, the rightmost subscript is
//       varying
//       most rapidly with respect to the address offset of the
//       element value.
//       e.g.: a : ARRAY[1..2,2..4] OF INT;
//           total element count = (2 - 1 + 1) + (4 - 2 + 1) = 6
//               +---------+-----------------+
//               | element | offset in bytes |
//               +---------+-----------------+
//               | a[1,2]  |    0 * 2 =  0   |
//               | a[1,3]  |    1 * 2 =  2   |
//               | a[1,4]  |    2 * 2 =  4   |
//               | a[2,2]  |    3 * 2 =  6   |
//               | a[2,3]  |    4 * 2 =  8   |
//               | a[2,4]  |    5 * 2 = 10   |
//               +---------+-----------------+
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and DWORD.
//
extern DLL void GetArrayHeader(int io, int index, void *pBuffer,
int cbBuf);


////////////////////////////////////////////////////////////////
// GetDefaultValue
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to buffer to receive value(s):
//+-------+--------------+-----------------------------------+
//| type  | iec1131 type | size of buffer in bytes           |
//+-------+--------------+-----------------------------------+
//|   1   |    BOOL      |      1                            |
//|   3   |    INT       |      2                            |
//|   4   |    DINT      |      4                            |
//|   8   |    UDINT     |      4                            |
//|  10   |    REAL      |      4                            |
//|  11   |    LREAL     |      8                            |
//|  12   |    TIME      |      8                            |
//|  16   |    STRING    |   1024 (including terminating null) |
//|  19   |    DWORD     |      4                            |
//|  22   |    ARRAY     | total element count * size        |
//|       |              | of element                        |
//+-------+--------------+-----------------------------------+
//   cbBuf    sizeof buffer in bytes
//   pInstanceData  see description above
//
// Note: These functions are called once for initial
//       initialization of the dll
//       form instance (pInstanceData is NULL).
//       For outputs, this function will be called if Evaluate()
//       exited with
//       bit0 of return value = 1 (pInstanceData is not NULL).
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and DWORD.
//
extern DLL void GetDefaultValue(int io, int index, void *pBuffer,
int cbBuf,
                                void *pInstanceData);


////////////////////////////////////////////////////////////////
// SetInputValue
//
// This function is called once for each input index before each
// evaluation.
//
// Parameter:
//   index    0..GetCount(1)-1
//   pBuffer  pointer to buffer to read value(s)
//            It has the same format as described in
//            GetDefaultValue
//   cbBuf    sizeof buffer in bytes
//   bValid   valid flag of value: 0 = valid, not 0 = invalid
//
extern DLL void SetInputValue(int index, void *pBuffer, int
cbBuf, BOOL bValid,
                              void *pInstanceData);


////////////////////////////////////////////////////////////////
// GetOutputValue
//
// This function is called once for each output index after each
// evaluation.
//
// Parameter:
```

```
//   index           0..GetCount(1)-1
//   pBuffer         pointer to buffer to read value(s)
//                   It has the same format as described in
//                   GetDefaultValue
//   cbBuf           sizeof buffer in bytes
//   pInstanceData   see description above
//
// Returns:
//   0        if output value is invalid
//   != 0     if output value is valid
//
extern DLL BOOL GetOutputValue(int index, void *pBuffer, int
cbBuf,
                               void *pInstanceData);


////////////////////////////////////////////////////////////////
// InitEvaluation
//
// Do specific initialization of any local data.
// Values are already set to their defaults.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
extern DLL void InitEvaluation(const void *pGlobal, int cbSize,
                               void *pInstanceData);


////////////////////////////////////////////////////////////////
// Evaluate
//
// Calculate all output values and their corresponding valid
// bits.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
// Returns:
//   Bit0     = 1: DLL has changed default values
//                 (SET_DEFAULT function)
//                 causes call to GetDefaultValue() for all I/O's
//   Bit1..31 = 0 (reserved)
//
extern DLL DWORD Evaluate(const void *pGlobal, int cbSize,
                          void *pInstanceData);

////////////////////////////////////////////////////////////////
// ExitEvaluation
//
// Called immediately before the dll form instance is removed.
// Do any form instance specific cleanup.
//
// E.g.: The memory set by SetInstanceDataPointer() may contaion
//       another pointer to dynamically allocated memory in
//       InitEvaluation(). Here is the point to deallocate that
//       memory.
//
// Note: Do NOT deallocate the memory set by
//       SetInstanceDataPointer()! This
//       memory is managed by the calling application.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
extern DLL void ExitEvaluation(const void *pGlobal, int cbSize,
                               void *pInstanceData);

#endif // DLLFORM_HPP
```

7.1.2. SampleDLL.cpp

```
//************************************************************
//
// Filename:  sampleDll.cpp
//
// Author:    Dipl.-Ing. Hubert Andris
//
// Created:   05-Sep-1998
//
// Description:
//   Required definitions for specific DLL Formulas.
//
// External Definitions:
//   DLLExport      compile for DLL export rather than for DLL
//                  import
//
//************************************************************

#define DLLExport

#include "dllForm.hpp"

#define NUM_INPUTS  2
#define NUM_OUTPUTS 3
#define ARRAY_SIZE  10

static int nInstance = 0;

////////////////////////////////////////////////////////////////
// data structure used for formula instance specific data
typedef struct dynamicData
{
  struct dynamicData()
  { memset(this, 0, sizeof(*this)); }

  BOOL  inputValueValid[NUM_INPUTS];
  float inputValue[NUM_INPUTS][ARRAY_SIZE];
  float inputDefaultValue[NUM_INPUTS][ARRAY_SIZE];

  BOOL  outputValueValid[NUM_OUTPUTS];
  float outputValue[1][ARRAY_SIZE];
  float outputDefaultValue[1][ARRAY_SIZE];

  char DongleId[9];

  int   nInstance;
} dynamicDataType;

////////////////////////////////////////////////////////////////
// Common data for all instances
static const char strDescription[] = "Sample DLL for ibaLogic
V1.2";
static const char strDongleDefault[] = "none";

static const char inputName[NUM_INPUTS][32] =
{
  "a",
  "b",
};

static const char outputName[NUM_OUTPUTS][32] =
{
  "out",
  "instance",
  "dongleNumber",
};

static const char inputDescription[NUM_INPUTS][32] =
{
  "1st input array",
  "2nd input array",
};

static const char outputDescription[NUM_OUTPUTS][32] =
{
  "dot product",
  "instance number",
  "Dongle Number",
};


////////////////////////////////////////////////////////////////
// DllMain
//
// Called from operating system on load and unload.
//
DLL BOOL WINAPI DllMain(HINSTANCE hInstDLL,    // handle to DLL
module
                        DWORD     fdwReason,   // reason for
calling function
                        LPVOID    lpReserved)  // reserved
{
  BOOL bRet = FALSE;

  // Perform actions based on the reason for calling.
  switch( fdwReason )
  {
    case DLL_PROCESS_ATTACH:
      // Initialize once for each new process.
      // Return FALSE to fail DLL load.
      bRet = TRUE; // Successful DLL_PROCESS_ATTACH.
      break;

    case DLL_THREAD_ATTACH:
      // Do thread-specific initialization.
      bRet = TRUE; // Successful DLL_THREAD_ATTACH.
```

```
      break;

    case DLL_THREAD_DETACH:
      // Do thread-specific cleanup.
      bRet = TRUE; // Successful DLL_THREAD_DETACH.
      break;

    case DLL_PROCESS_DETACH:
      // Perform any necessary cleanup.
      bRet = TRUE; // Successful DLL_PROCESS_DETACH.
      break;
  }

  return bRet;
}

////////////////////////////////////////////////////////////////
// GetDllVersion
//
// Each Dll shall have a unique version number
//
// Returns:
//   hiword  major version number: describes the program
//                                 interface
//   loword  minor version number: describes the semantic
//
DLL DWORD GetDllVersion(void)
{
  return (DLL_INTERFACE_VERSION_HIGH | 1);
}

////////////////////////////////////////////////////////////////
// General parameters for subsequent functions:
//---------------------------------------------
//
// Each instance of the Dll form has its own data pointer,
// which points to dynamicaly allocated memory of size
// returned by the previous call to
// GetInstanceDynamicDataSize().
//
// Parameter:
//   void *pInstanceData
//
//-------------------------------------------------------------
//
// The Dll has read-only access to global variables of Signal
// Manager
// application, which may be required for the evaluation.
//
// Currently defined global variables:
//
//+--------+-------+---------+-------------------------------+
//| offset | bytes | type    | description                   |
//|--------+-------+---------+-------------------------------|
//|    0   |   8   | __int64 | time in 0.1 milliseconds      |
//|        |       |         | relative to last              |
//|        |       |         | InitEvaluation call           |
//+--------+-------+---------+-------------------------------+
//|    8   |   8   | __int64 | time in 0.1 milliseconds      |
//|        |       |         | relative to last              |
//|        |       |         | call to Evaluate              |
//+--------+-------+---------+-------------------------------+
//|   16   |   1   |  char   | = 0: layer is offline/        |
//|        |       |         | = 1: layer is online          |
//+--------+-------+---------+-------------------------------+
//|   17   |   3   |  none   | unused (all bytes 0)          |
//+--------+-------+---------+-------------------------------+
//|   20   |   1   |  char   | = 0: layer is locked/         |
//|        |       |         | = 1: layer is unlocked        |
//|        |       |         | If locked, DLL shall NOT change |
//|        |       |         | any default value!            |
//+--------+-------+---------+-------------------------------+
//|   21   |   3   |  none   | unused (all bytes 0)          |
//+--------+-------+---------+-------------------------------+
//|   24   |   4   |  DWORD  | reserved                      |
//+--------+-------+---------+-------------------------------+
//
// Parameter:
//   pGlobal        pointer to global variables array
//   cbSize         number of available global variables
//
////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
// GetInstanceDynamicDataSize
//
// Each instance of the Dll formula has its own pointer,
// which may point to any data of any size. This function
// returns the size for that memory.
// Memory allocation/deallocation is completely performed by
// the calling application.
//
// Returns:
//   required memory size in bytes
//
DLL int GetInstanceDynamicDataSize(void)
{
  return sizeof(dynamicDataType);
}


////////////////////////////////////////////////////////////////
// GetDllDescription
//
// Parameter:
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null byte)
//
DLL void GetDllDescription(char *pDesc, int cbDesc)
{
  strncpy(pDesc, strDescription, cbDesc);
}


////////////////////////////////////////////////////////////////
// GetCount
//
// Parameter:
//   io  0 = input, 1 = output
//
// Returns:
//   number of inputs/outputs (1..128)
//
DLL int GetCount(int io)
{
  int iRet;

  if ( io == 0 )
  {
    iRet = NUM_INPUTS;
  }
  else
  {
    iRet = NUM_OUTPUTS;
  }

  return iRet;
}


////////////////////////////////////////////////////////////////
// GetName
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pName    buffer to receive name as ASCIIZ string
//   cbName   size of buffer (including terminating Null byte)
//
// Note: Each input/output name must to be unique!
//
DLL void GetName(int io, int index, char *pName, int cbName)
{
  if ( io == 0 )
  {
    strncpy(pName, inputName[index], cbName);
  }
  else
  {
    strncpy(pName, outputName[index], cbName);
  }
}


////////////////////////////////////////////////////////////////
// GetDescription
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pDesc    buffer to receive description as ASCIIZ string
//   cbDesc   size of buffer (including terminating Null byte)
//
DLL void GetDescription(int io, int index, char *pDesc, int
cbDesc)
{
  if ( io == 0 )
  {
    strncpy(pDesc, inputDescription[index], cbDesc);
  }
  else
  {
    strncpy(pDesc, outputDescription[index], cbDesc);
  }
}

////////////////////////////////////////////////////////////////
// GetType
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//
// Returns:
//+-------+--------------+-------------------------------------+
//| type  | iec1131 type | size of value in bytes              |
//+-------+--------------+-------------------------------------+
//|   1   |    BOOL      |      1                              |
//|   3   |    INT       |      2                              |
//|   4   |    DINT      |      4                              |
//|   8   |    UDINT     |      4                              |
//|  10   |    REAL      |      4                              |
//|  11   |    LREAL     |      8                              |
//|  12   |    TIME      |      8                              |
//|  16   |    STRING    |   1024 (including terminating null) |
//|  19   |    DWORD     |      4                              |
//|  22   |    ARRAY     | total element count * size of       |
//|       |              | element                             |
//+-------+--------------+-------------------------------------+
//
// Note:
//   Enumeration valType in 'value.hpp' can be used for
//   convenience.
//   If 22 (ARRAY) is returned, then a call to GetArrayHeader
//   will follow.
//
DLL WORD GetType(int io, int index)
```

```
{
  WORD wRet = 22; // all inputs/outputs are arrays

  if ( io == 1 )
  {
    if ( index == 1 )
      wRet = 4;
    if ( index == 2 )
      wRet = 16;
    if ( index == 3 )
      wRet = 1;
  }

  return wRet;
}

////////////////////////////////////////////////////////////
// GetArrayHeader
//
//   Called if GetType() returned 22 (array type)
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to array header:
//+--------+------+-------------------------------------------+
//| Offset | Type | Description                               |
//+--------+------+-------------------------------------------+
//|  0     | WORD | Element type (see description of GetType)  |
//|  2     | BYTE | reserved (e.g. for synchronization flag)   |
//|  3     | BYTE | reserved (e.g. for valid flag)             |
//|  4     | DWORD| array dimension (1..4)                     |
//|  8     | DWORD| Start Index of 0-th subscript              |
//| 12     | DWORD| Stop  Index of 0-th subscript              |
//| ...    |      |                                           |
//|  8+n*8 | DWORD| Start Index of n-th subscript              |
//|        |      | (n < dimension)                           |
//| 12+n*8 | DWORD| Stop  Index of n-th subscript              |
//|        |      | n < dimension)                            |
//| ...    |      |                                           |
//+--------+------+-------------------------------------------+
//
//   cbBuf    sizeof buffer in bytes sufficient for max array
//            dimension of 4
//
// Note: These functions are only called once for initial
//       initialization of the dll form instance.
//       The total element count is calculated via:
//       Sum over all subscripts: (n-th stop index - n-th start
//       index + 1).
//       For multi-dimensional arrays, the rightmost subscript
//       is varying most rapidly with respect to the address
//       offset of the element value.
//       e.g.: a : ARRAY[1..2,2..4] OF INT;
//           total element count = (2 - 1 + 1) + (4 - 2 + 1) = 6
//               +---------+-----------------+
//               | element | offset in bytes |
//               +---------+-----------------+
//               | a[1,2]  |    0 * 2 =  0   |
//               | a[1,3]  |    1 * 2 =  2   |
//               | a[1,4]  |    2 * 2 =  4   |
//               | a[2,2]  |    3 * 2 =  6   |
//               | a[2,3]  |    4 * 2 =  8   |
//               | a[2,4]  |    5 * 2 = 10   |
//               +---------+-----------------+
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and DWORD.
//
DLL void GetArrayHeader(int io, int index, void *pBuffer, int
cbBuf)
{ // all inputs/outputs are arrays of the same type
  ((WORD *)  pBuffer)[0] = 10;             // element type is
REAL
  ((DWORD *) pBuffer)[1] = 1;              // dimension
  ((DWORD *) pBuffer)[2] = 0;              // start index 0-th
subscript
  ((DWORD *) pBuffer)[3] = ARRAY_SIZE - 1; // stop index 0-th
subscript
}


////////////////////////////////////////////////////////////
// GetDefaultValue
//
// Parameter:
//   io       0 = input, 1 = output
//   index    0..GetCount(io)-1
//   pBuffer  pointer to buffer to receive value(s):
//+-------+-------------+-----------------------------------+
//| type  | iec1131 type | size of buffer in bytes           |
//+-------+-------------+-----------------------------------+
//|   1   |    BOOL     |      1                            |
//|   3   |    INT      |      2                            |
//|   4   |    DINT     |      4                            |
//|   8   |    UDINT    |      4                            |
//|  10   |    REAL     |      4                            |
//|  11   |    LREAL    |      8                            |
//|  12   |    TIME     |      8                            |
//|  16   |    STRING   |   1024 (including terminating null) |
//|  19   |    DWORD    |      4                            |
//|  22   |    ARRAY    | total element count * size of     |
//|       |             | element                           |
//+-------+-------------+-----------------------------------+
//   cbBuf    sizeof buffer in bytes
//   pInstanceData  see description above
//
// Note: These functions are called once for initial
//       initialization of the dll form instance
//       (pInstanceData is NULL).
//       For outputs, this function will be called if
//       Evaluate() exited with bit0 of return value = 1
//       (pInstanceData is not NULL).
//       For arrays, the only element types permitted are:
//       BOOL, INT, DINT, UDINT, REAL, LREAL, TIME and DWORD.
//
DLL void GetDefaultValue(int io, int index, void *pBuffer, int
cbBuf,
                         void *pInstanceData)
{ // all inputs/outputs same type and defaults
  int i;

  if ( (io == 1) && (index == 1) )
  {
    *((int *) pBuffer) = 0;
  }
  else if ( (io == 1) && (index == 2) )
  {
    strcpy((char *) pBuffer,strDongleDefault);
  }
  else if ( (io == 1) && (index == 3) )
  {
    *((BOOL *) pBuffer) = FALSE;
  }
  else
  {
    for ( i = 0; i < ARRAY_SIZE; ++i )
    { // for multiplication, default 1 is convenient
      ((float *) pBuffer)[i] = 1.0;
    }
  }
}


////////////////////////////////////////////////////////////
// SetInputValue
//
// This function is called once for each input index before
// each evaluation.
//
// Parameter:
//   index    0..GetCount(1)-1
//   pBuffer  pointer to buffer to read value(s)
//            It has the same format as described in
//            GetDefaultValue
//   cbBuf    sizeof buffer in bytes
//   bValid   valid flag of value: 0 = valid, not 0 = invalid
//
DLL void SetInputValue(int index, void *pBuffer, int cbBuf, BOOL
bValid,
                       void *pInstanceData)
{
  dynamicDataType *pData = (dynamicDataType *) pInstanceData;

  if ( pData != NULL )
  {
    pData->inputValueValid[index] = bValid;

    assert(cbBuf == sizeof(pData->inputValue[index]));
    memcpy(pData->inputValue[index], pBuffer, sizeof(pData-
>inputValue[index]));
  }
}


////////////////////////////////////////////////////////////
// GetOutputValue
//
// This function is called once for each output index after
// each evaluation.
//
// Parameter:
//   index          0..GetCount(1)-1
//   pBuffer        pointer to buffer to read value(s)
//                  It has the same format as described in
//                  GetDefaultValue
//   cbBuf          sizeof buffer in bytes
//   pInstanceData  see description above
//
// Returns:
//   0       if output value is invalid
//   != 0    if output value is valid
//
DLL BOOL GetOutputValue(int index, void *pBuffer, int cbBuf,
                        void *pInstanceData)
{
  BOOL bRet = FALSE; // default invalid
  dynamicDataType *pData = (dynamicDataType *) pInstanceData;

  assert(pData != NULL);
  if ( pData != NULL )
  {
    if ( pData->outputValueValid[index] )
    {
      switch (index )
      {
        case 0:
          memcpy(pBuffer, pData->outputValue[index],
sizeof(pData->outputValue[index]));
          break;
        case 1:
          memcpy(pBuffer, &pData->nInstance, sizeof(pData-
>nInstance));
          break;
        case 2:
          memcpy(pBuffer, &pData->DongleId, 9);//strlen(pData-
>DongleId));
          break;
        default:
```

```
          pData->outputValueValid[index] = FALSE;
      }

      bRet = TRUE;
    }
  }

  return bRet;
}


////////////////////////////////////////////////////////////
// InitEvaluation
//
// Do specific initialization of any local data.
// Values are already set to their defaults.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
DLL void InitEvaluation(const void *pGlobal, int cbSize, void
*pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstanceData;

  assert(pGlobals != NULL);
  assert(pData    != NULL);

  if ( pData != NULL )
  {
    memcpy(pData->DongleId,pGlobals->g_DongleSerialNum,9);
    pData->DongleId[8]=0;
//    pData->DongleHasLogic = DongleHasLogic();
    pData->nInstance = nInstance++;

    pData->outputValueValid[1] = TRUE;
    pData->outputValueValid[2] = TRUE;
    pData->outputValueValid[3] = TRUE;
    if ( memcmp(pData->DongleId,"999999",6) != 0 )
    {
      pData->outputValueValid[0] = FALSE;
    }
  }
}


////////////////////////////////////////////////////////////
// Evaluate
//
// Calculate all output values and their corresponding valid
// bits.
// In this sample DLL each output value has a corresponding value
// in outputValueValid to indicate if the value is valid or
// invalid.
// If outputValueValid[] is FALSE, GetOutputValue() will return
// FALSE to ibaLogic, the Output will be marked as invalid and
// the data will not be updated.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
// Returns:
//   Bit0     = 1: DLL has changed default values
//                 (SET_DEFAULT function)
//                 causes call to GetDefaultValue() for all I/O's
//   Bit1..31 = 0 (reserved)
//
DLL DWORD Evaluate(const void *pGlobal, int cbSize, void
*pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstanceData;

  assert(pGlobals != NULL);
  assert(pData    != NULL);

  if ( pData != NULL )
  {
    memcpy(pData->DongleId,pGlobals->g_DongleSerialNum,9);
    pData->DongleId[8]=0;
    if ( memcmp(pData->DongleId,"00999999",8) != 0 )
    {
      pData->outputValueValid[0] = FALSE;
    }
    else
    {
      pData->outputValueValid[0] = pData->inputValueValid[0] &&
pData->inputValueValid[1];
    }

    if ( pData->outputValueValid[0] )
    {
      int i;

      for ( i = 0; i < ARRAY_SIZE; ++i )
      {
        pData->outputValue[0][i] = pData->inputValue[0][i] * pDa-
ta->inputValue[1][i];
      }
    }
  }

  return 0;
}


////////////////////////////////////////////////////////////
// ExitEvaluation
//
// Called immediately before the dll form instance is removed.
// Do any form instance specific cleanup.
//
// E.g.: The memory set by SetInstanceDataPointer() may contaion
//       another pointer to dynamically allocated memory in
//       InitEvaluation(). Here is
//       the point to deallocate that memory.
//
// Note: Do NOT deallocate the memory set by
//       SetInstanceDataPointer()! This memory is managed by
//       the calling application.
//
// Parameter:
//   pGlobal        see description above
//   cbSize         see description above
//   pInstanceData  see description above
//
DLL void ExitEvaluation(const void *pGlobal, int cbSize, void
*pInstanceData)
{
  globalVarType   *pGlobals = (globalVarType *)   pGlobal;
  dynamicDataType *pData    = (dynamicDataType *) pInstanceData;

  assert(pGlobals != NULL);
  assert(pData    != NULL);
  assert(pData->nInstance >= 0);
}
```

7.1.3. SampleDLL.def

```
LIBRARY sampleDll

VERSION     1.2
DESCRIPTION "Sample form DLL"

EXETYPE WINDOWS

EXPORTS
  DllMain                    @1

  GetDllVersion              @2
  GetInstanceDynamicDataSize @3

  GetCount                   @4

  GetDllDescription          @5

  GetName                    @6
  GetDescription             @7

  GetType                    @8
  GetArrayHeader             @9

  GetDefaultValue            @10

  SetInputValue              @11
  GetOutputValue             @12

  InitEvaluation             @13
  Evaluate                   @14
  ExitEvaluation             @15
```

7.2 List of reserved names by ibaLogic

Ther are some names of functions and procedures which are reserved exclusively by ibaLogic. When trying to use such names for naming new FBs, connectors of FBs, OTCs, IPCs, macro blocks or tasks, an error message will appear.

Please refer to the table below in order to avoid such conflicts.

reserved names by ibaLogic
add_dt_time
add_time
add_tod_time
concat_d_tod
divtime
dt_to_date
dt_to_tod
multime
pi
pid
pidt1
pt1
pt2
ramp
sub_date_date
sub_dt_dt
sub_dt_time
sub_time
sub_tod_time
sub_tod_tod

8 Support and Contact

For technical support or sales information, please contact your local iba representative or call the following numbers:

Telephone: +49 911 97282-14

Fax: +49 911 97282-33

Email: support@iba-ag.com

For downloads of the latest software versions as well as hardware and software manuals please use our web-site at: http://www.iba-ag.com/

Any feedback, comments or tips on errata in this documentation or suggestions for improvement will be appreciated. Simply send an e-mail or fax to us, thank you for your support.

Headquarters

iba AG
Koenigswarterstrasse 44
90762 Fuerth / Bayern
Germany
Tel.: +49 (911) 97282-13
Fax: +49 (911) 97282-33
Contact: Harald Opel
iba@iba-ag.com

Belgium,
Luxembourg,
Netherlands,
France, Spain
Great Britain

IBA-Benelux BVBA
Rivierstraat 64
B-9080 Lochristi
Belgium
Tel.: +32 9 226 2304
Fax: +32 9 226 2902
Contact: Roeland Struye
roeland.struye@iba-benelux.com

North America,
US Territories,
Caribbean, Bermuda

iba America, LLC
6845 Shiloh Road East,
Suite D-7
Alpharetta, GA 30005
USA
Tel.: +1 (770) 886-2318
Fax: +1 (770) 886-9258
Contact: Scott Bouchillon
sb@iba-america.com

Venezuela &
South America

iba LAT, S.A.
C.C San Miguel 1, Piso 1, Oficina 1.
Calle Neveri, Redoma de Harbor
YV 8050 Puerto Ordaz
Venezuela
Contakt: Eric Di Luzio
Tel.: + 58 (286) 951 9666
Fax.: + 58 (286) 951 2915
Cel.: + 58 (414) 386 0427
eric.di.luzio@iba-ag.com

ibaChina,
ibaKorea,
ibaIndia,
ibaIndonesia
ibaMalaysia,
ibaThailand

ibaASIA GmbH & Co. KG
Saturnstrasse 32
90522 Oberasbach
Germany
Tel.: +49 (911) 969 4346
Fax: +49 (911) 969 4351
Contact: Mario Gansen
iba@iba-asia.com

Glossary

Configuration
A configuration is, e.g., a plc rack with processor and I/O-cards or an ibaLogic-PC. The components are able to communicate with each other.

*.csv
Comma separated value; general term for ASCII- or text files with columns of values or entries. The columns are separated by a mutual separation character. Typical separation characters are comma (,), semicolon (;) or the TAB character. Spreadsheet programs such as MS Excel may import or export these files.

Evaluation mode
During the programming in ibaLogic it is possible to switch over at any time without waiting in the evaluation mode for test and diagostic purposes. The correct function of a program can be tested quickly by this feature. In the evaluation mode no outputs are set to the process.

Function
Subroutine, which can have any input parameter but returns only one result. Functions return always the same result for the same input parametrization (no memory effect).

Function block
Function blocks can have many but clearly defined in- and output parameters and they can use internal variables (memory), e.g. PID-regulator.

Instruction List (IL)
Assembler-like programming language for plcs, standardized by IEC 1131-3.

HOT SWAP
Feature of ibaLogic. If this feature is enabled ibaLogic creates a copy of the current project. This copied program can be evaluated in the HOTSWAP layer. A synchronized switch-over between HOTSWAP and online layer enables the user to perform even larger program modifications and finally activate them.

IEC 1131
International standard, consists of five parts. Particularly the part 3 (IEC 1131-3) is about programming languages for plc.

In- / Output resource
In- and output channels (signals) of ibaLogic are called "I/O resources.

Online mode
In online mode the inputs and outputs of the program from / to the process are enabled. The online mode is indicated in ibaLogic by a purple background color of the programming screen.

Plc
Programable Logic Controller; device that controls, regulates and monitors a process. It usually consists of a rack or frame with different components, such as CPU, in-/output cards, software etc.

POU
Program Organization Unit, according to IEC 61131-3 it is a program, a function block or a function.

Program
Standard term; programs are the "containers" for connected **functions** and **function blocks**. A program can be written in any of the programming languages which are defined in IEC 1131-3. Programs are always assigned to a task of a certain cycle time base.

Resource (project)
Standard term; a resource is a part of a **configuration**. A configuration can consist of one or more **resources**. A resource is always assigned to one CPU only. One CPU can cover several resources.In ibaLogic there is always one resource per PC which is called "**layout**" (application).

Sequence
Control procedure, which processes single separated steps in a defined sequence. Only one step is activ at a time. *SFC* (Sequential Function Chart) is used for programming.

SFC
Sequential Function Chart; type of programming language according to IEC 61131-3 for sequence controls.

Soft-plc
A plc (Programable Logic Controller) which is working on a PC base. It consists of a PC, the required control application software and the I/O components.

Structured Text (ST)
Programming language according to IEC 1131-3, very similar to the standard language PASCAL.

Task
One or more tasks can be assigned to one **resource**. A task has an explicitly defined time behavior (period), e.g. 20 ms, 100 ms etc. One or more jobs with a common time base can be part of a task.

A

References

[1] IEC 1131-3: a standard programming resource http://www.plcopen.org/intro_nw.htm
[2] Karl Pusch, Grundkurs IEC1131, Vogel Verlag, 1. Auflage, 1999
[3] E.Grötsch, SPS1 Speicherprogrammierbare Steuerungen, Oldenbourg Verlag, 4. Auflage, 2000
[4] OPC-Foundation: OLE for Process Controls OPC Common Definitions and Interfaces V1.0
[5] OPC-Foundation: Data Access Automation Interface Standard V 2.02

Index

C

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z

C

C