STNASPROG-E

B&R AUTOMATION STUDIO™
PROGRAMMING

Model No.: STNASPROG-E

Version: 2.0
MP/SP/ZR 04/2001
SEMINAR START

1 INTRODUCTION .................................................................................. 2

2 SEMINAR OVERVIEW ........................................................................ 3

3 SCHEDULE ....................................................................................... 4
1 INTRODUCTION

General Information
In the next few days - for the duration of this seminar - you will be working together with your seminar leader. The technical expertise of your seminar leader is not the only factor responsible for your personal success during this seminar.

Success depends on cooperation and interaction between the course members and the seminar leader, as well as the attitude and application of individual course members towards teamwork and the course in general.

Introduction Seminar Leader
Please allow the seminar leader to introduce him/herself. Make notes if necessary.

Introduction Course Members
You should get to know your colleagues as you will be working together as a group and also in smaller groups during the seminar (name, company, product, application area).

Your personal success on this course also depends on your expectations.
2 SEMINAR OVERVIEW

- B&R Automation Studio
- B&R Automation Runtime
- B&R Automation Target
- B&R Automation Net
- Project Guidelines
- SFC Seqential Function Chart
- AB Automation Basic
- Data Handling
- TPU Code Linker
- Library Manager Introduction
- ANSI C
3 SCHEDULE

The time available during the seminar is a very important factor.

Start

Lunch Break

End

Breaks  We will taking short breaks at various intervals through out the seminar for tea and coffee and to give the smokers the chance to light up!

Successful teamwork not only relies on your personal motivation, but also on meeting the expectations of your seminar leader.

The seminar leader expects:

That all course members are prepared to take an active role and cooperate with other course members during the course of the seminar. “Nobody is perfect”, this includes your trainer, constructive feedback is always welcome.
B&R AUTOMATION STUDIO

1 OVERVIEW ................................................................. 2

2 B&R AUTOMATION STUDIO ........................................ 3
   2.1 One Tool Many Targets ........................................... 3

3 AS DIRECTORY STRUCTURE ...................................... 4

4 PROJECT DIRECTORY STRUCTURE .............................. 5

5 AS PROJECT ............................................................... 6
   5.1 Opening Projects .................................................... 6
   5.2 LAD Task ............................................................... 8
   5.3 C Task ................................................................. 9
   5.4 Variable Declaration .............................................. 10
   5.5 Transferring Projects ............................................ 11

6. PROJECT SETTINGS ................................................... 13
   6.1 IEC Editor ............................................................ 14
   6.2 Ladder Printer Settings ......................................... 15
   6.3 C Compiler .......................................................... 16
   6.4 Build and Transfer ............................................... 17
   6.5 Motion ................................................................. 18
   6.6 Library Manager .................................................... 19
   6.7 Naming Conventions ............................................ 20
1 OVERVIEW

During the ASINT course, we got to know Automation Studio as a general tool. Our task in this course is to learn about AS features in greater detail, which will help you use the full range of advantages for many different types of applications.

B&R Automation Studio

AS Directory Structure
The question: “Where can I find it?” will be answered in this chapter. Finding header files, archive files, system files, etc.

Project Directory Structure
The question: “Where should I save it?” is answered in this chapter. Storing source files, executable BR files, project info files, etc.

AS Project
Brief description of the AS project structure and an overview of the most important points from creation to downloading a project.

Project Settings
Overview of the global project parameters.
2 B&R AUTOMATION STUDIO

2.1 One Tool Many Targets

There is a programming tool, B&R Automation Studio, that can be used with many different target systems. This enables simple scaling and optimal compatibility with the automation platform. B&R Automation Studio communicates via B&R Automation Net using B&R Automation Runtime. B&R Automation Runtime runs on various B&R Automation targets.
3 AS DIRECTORY STRUCTURE

You can find your way around the AutomationStudio structure quickly and easily using the following directory list which provides an overview of the directory contents. This structure is set up when you install Automation Studio.

![Directory Structure Diagram]

Fig. 2.1: B&R Automation Studio directories
4 PROJECT DIRECTORY STRUCTURE

When you create a new project with AutomationStudio, it automatically creates a project structure and gives it the directory name: “Project name.pgp”.

All project specific data, information, sources, etc. are stored in a directory structure under this directory. This allows you to move or copy the project directory at a later date, and to make it easily accessible for other programmers.

The project directory has the following structure:

![Diagram of AS project directory structure]

Note

The last two directories “inc” and “src” can be created by the user if C tasks are used.
5 AS PROJECT

5.1 Opening Projects

In order to be able to open an existing project, select the main menu item **File: Open Project.**

Select the respective GDM file, e.g.: “RIO_2005.GDM” and activate it with [Open]. GDM stands for Graphic Design Method.

This GDM file is in the project directory and has the name “prj_name.pgp”.

![Open Project dialog box](image)
A project is an entire system or machine and can be separated in the hardware configuration into Project, PCC, and CPU layers. On the software page, the CPU is separated into the individual task classes with their tasks and system modules.

This has a tree structure in the project window.

Fig. 2.4: Overview of Automation Studio project

- a ... Title bar with name of the open project
- b ... Menu bar
- c ... Toolbar
- d ... Root directory of the project tree is the project name
- e ... Controller name, PCC name
- f ... The root directory of the software tree is the selected CPU
- g ... Task classes with defined cycle time
- h ... LAD Task
- i ... C task, can consist of several files
- k ... C task source file
- m ... System modules
5.2 LAD Task

A new task can be created using the Object Wizard by selecting **Insert: New Object: LAD Task**. The source code for a LAD task (*.SRC extension) is always placed in the CPU program module directory:

![Fig. 2.5: New LAD task in the software tree](image1)

![Fig. 2.6: Source code in the project structure](image2)
5.3 C Task

Using **Insert: New Project: C Task**. When inserting a C task, the C files are normally copied to the CPU program module directory (*.c/*.h/*.s/*.a/*.o). Save the C files in a subdirectory to improve clarity of the project.

Also see **Project Guidelines II** or chapter **ANSI C**.
5.4 Variable Declaration

The menu **Open: Declaration** opens the respective variable declaration. If the CPU is selected, all global variables are shown.

![CPU variable declaration pop-up dialog box](image)

**Fig. 2.9: CPU variable declaration pop-up dialog box**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Attribute</th>
<th>Value</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>gPNIC_Pump10</td>
<td>GOL</td>
<td>global</td>
<td>PS1.1.11</td>
<td></td>
<td>Time switching delay</td>
</tr>
<tr>
<td>gPNIC_Pump10</td>
<td>GOL</td>
<td>global</td>
<td>PS1.1.1.2</td>
<td></td>
<td>Time switching delay</td>
</tr>
<tr>
<td>gPNIC_Pump1</td>
<td>GOL</td>
<td>global</td>
<td>PS1.1.2.1</td>
<td>remnant</td>
<td>Transistor 2A/24VDC</td>
</tr>
</tbody>
</table>

**Fig. 2.10: All global variables**

- **Name**
  All variables are accessed using their symbolic names.

- **Type**
  Types are automatically assigned for I/O. The programmer has to enter the type for internal variables used in tasks.

- **Scope**
  Internal variables should always have the smallest possible scope. I/O and variables for communication between task must be global.

- **Attribute**
  Defines hardware assignments, internal variables or constants.

- **Owner**
  The owner e.g.: of a L structure or constant is shown.

- **Value**
  Initialization value. Should be remnant. Variables should be initialized in the INIT SP for the task.

- **Remark**
  Additional information or standard text.
5.5 Transferring Projects

The main menu item **Project: Transfer to Target** transfers all the software modules to the controller. An automatic software compare is carried out between the PCC and the project before the transfer is made. Only modules currently not available or with newer creation dates are transferred to the controller.

The target memory for each individual module can be selected by marking the object with the cursor and selecting the main menu item **Object: Transfer to: Target Memory**.

If AS finds one or more differences between the project contents and the PCC, it informs the programmer with the info message “Software Conflict” and displays the following dialog box.

**Note**

**Software Conflict Recognition**

- **Delete Object from the target**
  The module identified as being different is deleted from the controller.

**IMPORTANT**

modules identified as “disabled” are also deleted from the controller.

- **Ignore Object on the target**
  The module is not changed. If it is a task with I/O operations, conflicts can occur.
Example

- Create a LAD task with a latch.
- Transfer the project and test the task.
- Rename the task and transfer the project again.

Original Task:

Project Name: as_rev
Ladder Diagram Name: re_latch
Resource: C#2
Target memory: FLASH PROM

Changed Task:

Project Name: as_rev
Ladder Diagram Name: wd_new
Resource: C#2
Target memory: RAM
6. PROJECT SETTINGS

The parameters described in the following sections are globally valid for the current project.

The project settings can be found under **Project: Settings**.

![Fig. 2.12: Opening project settings](image)
6.1 IEC Editor

- **Autodeclaration**
  If new variables are entered, the variable declaration for these variables is automatically opened when Autodeclaration is activated.

- **Autoformat**
  The source code is automatically formatted, indentation and colored highlighting of keywords.

- **Tab-Width**
  Width of the tabulator.

- **Font**
  Font to be used in the editors.

- **Mark**
  Representation of marked text.

- **Monitoring options**
  If monitor mode is active, the editor window is split in two sections. The right section of the window shows the variables and their values. With „Width of monitor window“, you can set the width of the right section of the window as a percent of the total width of the window. The value „Distance of two variables“ determines the distance between two variables in the right section of the window if several variables are shown in a line.

- **Colors**
  You can define different colors for clear representation of line numbers, breakpoints, etc.
6.2 Ladder Printer Settings

- Symbol Details
  Which details should be printed for symbols.
- Print Order
  Order of the printout if the LAD does not fit on a page, is either too wide or too long.
- Symbol Print Size
  Size of the symbols in millimeter

Fig. 2.14: Ladder Print Settings
6.3 C Compiler

- Additional build options
  Information concerning options can be found in the GNU compiler documentation
- Include directories
  The list contains all include directories used by the compiler.

Fig. 2.15: C Compiler
6.4 Build and Transfer

- Generate Code for PP
  All PP specific compiler options are used.
- Locater local variables at
  variable area = DPR or USR RAM
- ASCII import
  Options for ASCII import
- Transfer
  Do not transfer library information to target system. The library information contains descriptions for calling FBKs which are only required for project creation. This information is not needed on the target system.
6.5 Motion

- **NC language**
  Language of the NC structures (English or German).
6.6 Library Manager

- Library directories
  List of directories to be used when searching for libraries.
- Standard directory
  The target memory for the libraries on the target system can be set here.

![Fig. 2.18: Library Manager](image-url)
6.7 Naming Conventions

- Identifiers
  Rules only according to the IEC standard or with B&R extensions.

Fig. 2.19: Naming conventions
# B&R AUTOMATION RUNTIME

1 OVERVIEW ............................................................................................... 2

2 AUTOMATION RUNTIME ........................................................................... 3

3 INDIVIDUAL SYSTEM CONFIGURATIONS .................................... 5
   3.1 Memory Configuration ..................................................................... 6
   3.2 Software Object Configuration ..................................................... 7
   3.3 System Configuration ..................................................................... 8
   3.4 Communication Configuration .................................................... 9
   3.5 Interfaces .................................................................................... 10
   3.6 Timing Configuration ..................................................................... 11
   3.7 Resource Configuration ............................................................ 12

4 ERROR LOGBOOK ............................................................................. 13

5 ONLINE INFORMATION ........................................................................ 15
   5.1 System Information ................................................................. 15
   5.2 CPU Memory Information ......................................................... 15
   5.3 Real-time Clock .......................................................................... 16

6 I/O HANDLING AND TIMING .......................................................... 17
   6.1 I/O Image Handling .................................................................... 18

7 SYSTEM STRENGTHS .......................................................................... 20
   7.1 Strengths of the B&R Multitasking System .................................. 20
   7.2 Strengths of the B&R I/O System ................................................. 21
1 OVERVIEW

B&R Automation Runtime
B&R Automation Runtime represents the layer between B&R Automation Studio and the B&R Automation Target.

Individual System Configuration
The PCC system is preconfigured by B&R. The task class times and memory sizes offered can be easily configured by the user. However, further adaptations should only be made by experienced users under B&R’s guidance. Also see online help.

Error Logbook
System messages are automatically entered in the error logbook. They can also be entered by an application using standard functions.

Online Information
General system, time and memory information can be requested and changed here.

I/O Timing
B&R provides customers with a big advantage, all systems can be programmed in the same way with the same program. This makes it easy for the user to make changes within the B&R2000 Family.

System Strengths
B&R Highlights
2 AUTOMATION RUNTIME

B&R Automation Runtime represents the layer between B&R Automation Studio and the B&R Automation Target. The connection is established via B&R Automation Net. The physical media used for communication is not important.
## B&R2000 CPU

### Functions

<table>
<thead>
<tr>
<th>OPSYS KERNEL</th>
<th>SYSCONF</th>
<th>RIOTRAP</th>
<th>PB_LIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXERMO</td>
<td>BURTRAP</td>
<td>NET2000</td>
<td>FRMDRV</td>
</tr>
</tbody>
</table>

## User / Application

<table>
<thead>
<tr>
<th>Communication</th>
<th>IO-Err Runtime</th>
<th>2010 (IO Bus) DI400</th>
<th>3 msec</th>
<th>2010 / 2005 RIO-SYNC</th>
<th>2010 HW-Timer</th>
<th>10 msec</th>
<th>2010 HW-Timer</th>
<th>9 msec</th>
<th>10 msec (20 msec)</th>
<th>50 msec (50 msec)</th>
<th>100 msec (100 msec)</th>
<th>10 msec (30.000 msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXC</td>
<td>IRQ</td>
<td>HS#1</td>
<td>HS#2</td>
<td>HS#3</td>
<td>TC#1</td>
<td>TC#2</td>
<td>TC#3</td>
<td>TC#4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Data

<table>
<thead>
<tr>
<th>Permanent</th>
<th>Remnant</th>
<th>GLOBAL VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>INPUT</td>
<td>INPUT</td>
</tr>
<tr>
<td>OUTPUT IMAGE</td>
<td>FIXRAM</td>
<td>DATA MODULE (Read/Write)</td>
</tr>
<tr>
<td>(Read only)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 INDIVIDUAL SYSTEM CONFIGURATIONS

Each user has the possibility with Automation Studio to easily configure the PCC according to individual requirements:

- Memory, e.g.: PCC global variable pool, digital and analog, FIX RAM, temporary memory
- Stack, e.g.: Task class stack C#1..C#4, operating system data stack
- Task class timing, e.g.: Duration and tolerance, idle time

We normally work with the standard settings. But if it is necessary to change one of the items listed above, make the change in the Object Properties Edit: Properties.

Note

Here, we also recommend using the shortcut menu by clicking on the object with the right mouse button!

Now we will discuss some important configurations. Information concerning the other configurations can be found in the online help.

If you lose track of the changes you have made, click on the [Default] button to reset the defaults.
3.1 Memory Configuration

- **Analog**
  PCC global PVs > Bit

- **Digital**
  PCC global PVs, Bit

- **Analog/Digital permanent**
  Permanent memory = cold restart safe.

- **FIX RAM**
  Cold restart safe RAM memory. The size is a multiple of 16 kByte.

- **TMP RAM**
  Temporary RAM memory. Initialized with 0 during each warm restart.
3.2 Software Object Configuration

- **Cyclic objects**
  Cyclic objects are application tasks.

- **User non-cyclic objects**
  are required for special system expansions. Can only be created together with B&R.

- **Sys. non-cyclic objects**
  Operating system expansions.

- **B&R objects**
  all modules that can be transferred to the controller, e.g.: tasks, data modules, etc.

- **Logbook size**
  certain system messages are also saved in the system logbook.
3.3 System Configuration

- **Queues**
  are used for communication between tasks.

- **Semaphores**
  are variables which can be used to synchronize access of common memory areas.

- **AVT entries**
  Address distribution tables. Used for library functions.

- **Stack**
  Operating system data memory. (automatically set starting with AR 2.22)

- **System modules**
  Number of hardware system modules.

- **Reboot mode after fatal error**
  If a fatal error occurs when starting the system, the system goes into diagnostic mode. That means the application is stopped. However, if “cold restart” is set, the system first attempts a cold restart. Then the application can continue running. If the error is still there, the system boots again in diagnose mode.
3.4 Communication Configuration

Communication channels
Corresponds to the number of interfaces that should be used at the same time for communication.

Device driver
Drivers for the various protocols (Net2000, Frame Driver, etc.)

**Force commissions**
Number of force commissions that can be active at the same time.

PV tables
A table contains max. 256 variables

Logical variable lists
are used for network communication.

Physical variable lists
are used for online communication

Connections
The number of parallel connections (INA) allowed can be set here.

Turbo Mode (starting with AR 2.20)
When activated, the INA services are also handled during idle time.
That means faster transfer of tasks, faster online communication, etc.
However, this mode only makes sense if enough idle time is available.
3.5 Interfaces

As default, AutomationRuntime defines the **first three interfaces** on the local CP for communication with AutomationStudio.

If additional interfaces are required for online communication with AutomationStudio, or special parameters are needed, they can be configured as parameter sets in this dialog box.

- **Interface**
  Several interfaces can be available on an interface module. These interfaces are numbered. The desired interface is selected using this parameter.

- **Type**
  Entry for the physical connection, e.g.: RS232, CAN, ETHERNET, etc.

- **Slot**
  Entry for the module slot number where the selected interface is found.

- **Subslot**
  Entry for the subslot on the module where the desired interface is found.

- **Parameter**
  Each interface type has default parameter. Changes can be made here, e.g.: Baudrate.

- **Modem Parameter**
  Special modem settings.
### 3.6 Timing Configuration

**Delay after cold/warm restart**
These times are delays that must pass before I/O modules or system modules can be accessed. In this way, these modules can complete their initialization before being accessed by the CPU.

**Power failure reporting delay**
After a power failure, the CPU executes a warm restart. For applications with expansion, it is possible that the expansion drops out before the CPU and the CPU activates service mode. When the voltage returns, the CPU starts in service mode. If a time is entered here, the CPU waits before activating service mode.

**Allowed system time violation**
The operating system is also monitored for cycle time violations. A maximum system cycle time violation can be set here, in 10ms steps.
3.7 Resource Configuration

![System Software Properties](image)

Fig. 3.9: Resource properties

- No. of cyclic resources
  Number of cyclic task classes possible.

- No. of timer resources
  Number of timer task classes.

- **Enable EXC class**
  Enable the exception task classes.

- Enable IRQ class
  Enable interrupt task classes, only possible on B&R2010.
4 ERROR LOGBOOK

All errors that trigger an EXCEPTION are entered in the system logbook. If an exception is triggered, all outputs are switched off. The application on the PCC is stopped. To determine the cause of the error, we will select:

- The CPU in the hardware tree
- The tab “Logbook” in the software tree

Then a window is opened which shows the last messages in plain text. The top message is the newest.

Some user actions are entered in the logbook, but the application continues running, e.g.: changing the time. Such entries are called WARNINGS.

<table>
<thead>
<tr>
<th>Time</th>
<th>Error</th>
<th>Information</th>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.12.98 15:45:23.00</td>
<td>2075</td>
<td>16#00000000</td>
<td>Syss</td>
<td>Warning: Time/date changed</td>
</tr>
</tbody>
</table>

Tab. 3.1: System entry – changed CPU time

If the operator exceeds defined limit values, a user entry can be created using a function and the PCC boots in service mode.

<table>
<thead>
<tr>
<th>Time</th>
<th>Error</th>
<th>Information</th>
<th>Module</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>???.?? ???.????.?</td>
<td>2222</td>
<td>16#12345678</td>
<td>???.?</td>
<td>Warning: ??????????</td>
</tr>
</tbody>
</table>

Tab. 3.2: User entry in logbook

In this case, the service technician must have a reference list of these entries. An action list is to be added to this reference list.
Example

Read the last entries from the system logbook and analyse the entries with your trainer.

- Read the entries

- Remove an I/O module accessed by the task with power applied. Then read the system logbook again.
5 ONLINE INFORMATION

5.1 System Information

If you click on the CPU symbol with the right mouse button in the hardware tree, the following dialog box is shown.

In the System dialog box, information concerning the boot mode, battery, operating system and node number are shown if available.

5.2 CPU Memory Information

Information concerning free memory is output here.
5.3 Real-time Clock

The time can be changed on the controller in the Date/Time dialog box. This dialog box shows the current time on the controller. This is not necessarily the same as the time on PC.

With [Get PC time], the current PC time is set in the dialog box. The settings can transferred to the target using [Set Target time] and the time change is entered in the system logbook.

![Fig. 3.13: CPU Info Date/Time properties](image-url)
6 I/O HANDLING AND TIMING

We will discuss the IO Handling and Timing here. The following points will be covered:

- I/O Image Handling
- Timing B&R2003
- Main Rack
  - CAN I/O
  - RIO
- Timing B&R2005 / B&R2010
  - B&R2005 Main Rack
  - B&R2010 Main Rack, B&R2005 and B&R2010 Expansion
6.1 I/O Image Handling

Input images are read separately for each task class. All task classes have a common **output image**.

Several tasks are executed in two task classes on a PCC:

The representations above show the following points clearly:

- The respective input images are read at the beginning of the task class.
- The output image is written at the end of run time for the task class.
- Differences between timing with and without an I/O processor.
6.1.1 B&R2003 / B&R2005 Main Rack

In this case, the main processor has to emulate the I/O processor. The main processor processes the system manager, then emulates the I/O processor and reads the inputs.

After this, the task class is processed.

During run time, the main processor emulates the I/O processor again and writes the outputs. The main processor is also responsible for I/O transfer.

6.1.2 B&R2005 Expansion and Remote I/O, B&R2010 with I/O Processor

This I/O processor reads the inputs parallel to system manager processing and writes the outputs at the end of the task class parallel to processing the next task class.

This multiprocessor concept in the CPU reduces load on the main processor caused by data transfer.

The I/O processor is responsible for I/O transfer. The remote IO master is responsible for remote IO transfer.

Note

If exact I/O times are required for an application, the exact information can be calculated by referring to the user’s manual or using an Excel file.
7 SYSTEM STRENGTHS

7.1 Strengths of the B&R Multitasking System

- **Deterministic Multitasking**
  Predictable task timing

- **Different Task Classes**
  Task classes are called in different, fixed time axes. Additional tasks do not change the timing.

- **Variable Task Class Cycle Times**
  Optimal timing settings for the different task classes.

- **Optimal Processor Load**

- **Priorities for Task Classes**
  Timer task class 1 has the highest priority of the cyclic task classes.

- **Flexible System Software Updates for the Operating System**

- **System Logbook**
  In the system logbook, an entry is always written describing the cause for a system reset. This logbook can also be used by the programmer, to check certain defined limits. If a limit is exceeded, an entry is made. In this way, the cause of a system standstill can always be determined.

- **Testing Individual Tasks**
  Start and stop tasks online, etc.
7.2 Strengths of the B&R I/O System

- **Separation of the I/O Bus**
  The I/O bus can be built with a decentralized structure using expansions and remote I/O.

- **Secure Protocol for I/O Bus**
  I/O data is transferred using a secure protocol and stored in the DPR.

- **Reducing Load on the CPU**
  I/O data transfer is handled by the I/O processor (B&R2010). Bit and byte masking is carried out by the DPR controller. Servicing the interfaces is carried out by the RISC. In this way, the maximum CPU capacity is available for application tasks.

- **Consistent I/O Images**
  The input states available at the beginning of the task class and remain available for the entire task class.

- **Symbolic Process Variables**
  I/O points are accessed in the program using symbolic names. The link between symbolic names and the I/O points can be defined at any time:
  - Taken from CAD programs
  - Predefined during programming
  - When creating code in LAD, C files, or when compiling
B&R AUTOMATION TARGETS

1 OVERVIEW ................................................................................................. 2

2 B&R AUTOMATION TARGET ........................................................................ 3

3 B&R AUTOMATION TARGET 2003 .............................................................. 4
   3.1 Main Unit ............................................................................................. 4
   3.2 Expansion ........................................................................................... 6
   3.3 CAN I/O Projects .............................................................................. 7

4 B&R AUTOMATION TARGET 2005 ............................................................ 12
   4.1 Main Unit ........................................................................................... 12
   4.2 Expansion ........................................................................................... 14
   4.3 RIO Projects ..................................................................................... 15

5 B&R AUTOMATION TARGET 2010 ........................................................... 20
   5.1 Main Unit ........................................................................................... 20
   5.2 Expansion ........................................................................................... 23

6 B&R AUTOMATION TARGET LOGIC SCANNER .................................... 26
   6.1 Main Unit ........................................................................................... 26
   6.2 Expansion ........................................................................................... 27

7 B&R AUTOMATION TARGET IPC2XXX .................................................... 28
   7.1 Main Unit ........................................................................................... 28
   7.2 Expansion ........................................................................................... 29

8 B&R AUTOMATION TARGET IPC5XXX .................................................... 30
   8.1 Main Unit ........................................................................................... 30
   8.2 Expansion ........................................................................................... 31

9 B&R2000 OVERVIEW .................................................................................. 32

Training A$PROG
1 OVERVIEW

B&R Automation Target

B&R Automation Target 2003
The most important data for the B&R2003 PCC System. Hardware possibilities, expansion possibilities with RIO and CAN I/O.

B&R Automation Target 2005
The most important data for the B&R2005 PCC System. Hardware possibilities, expansion possibilities with expansion and RIO.

B&R Automation Target 2010
The most important data for the B&R2010 PCC System. Hardware possibilities, expansion possibilities with expansion and RIO.

B&R Automation Target Logic Scanner
The combination of an IPC and LS251 results in a very powerful device for visualization and control tasks.

B&R Automation Target IPC2xxx
The most important data and hardware possibilities.

B&R Automation Target IPC5xxx
The most important data and hardware possibilities.
2 B&R AUTOMATION TARGET

B&R Automation Target refers to the **hardware platform**, where B&R Automation Runtime is running. This can be a B&R2003, B&R2005, B&R2010, IPC with Logic Scanner, IPC with B&R Automation Runtime (AR010).

Possible Automation Targets:
3 B&R AUTOMATION TARGET 2003

3.1 Main Unit

- The B&R2003 is used as an intelligent terminal because of its structure and connection technology.
- You can select between the intelligent PCC version and the unintelligent remote I/O version which reduces cabling costs.
- The special terminal blocks, which have separate terminals for signal, ground and supply, allows fast and easy cabling.
- The combination of different screw-in modules on a CPU or analog interface module guarantees the highest degree of modularity and the smallest size.

---

**CPU**

**Interface**

CPU I/O bus with secure data transfer; max. 8 logic modules

![Module rack + CPU with CP interface](image1)

*Fig. 4.1: Module rack + CPU with CP interface*

---

**CPU**

I/O bus with secure data transfer; max. 8 logic modules

![Module rack + CPU](image2)

*Fig. 4.2: Module rack + CPU*
3.1.1 Power Supply

CPU / CAN Bus Slave Module / RIO Slave Module
Supply voltages in the DC range from 18-30V and in the AC range from 82-264V, 47-63Hz. The power supplies are protected on the primary side by a fuse and on the secondary side by an internal current limiter.

3.1.2 Operating System

All B&R Automation Targets are compatible with regard to operating system functions and programming. A modem capable RS232 and a CAN interface are available.

3.1.3 Online

The first three interfaces can be used as online interface by the operating as default.

3.1.4 Application Program Memory

The application memory is on board. SRAM and FPROM are available as memory media. The SRAM is used for program development. The FIXRAM as part of SRAM for data that must remain after a cold start. The FPROM is used to store the operating system and completely tested projects including documentation.

3.1.5 I/O Modules and Terminals

The B&R2003 system offers a palette of

- **Digital Modules** (Relay or transistor version)
- **Analog Modules**
  - 0-20mA and ±10V ... Resolution: 12Bit
  - PT100 / FeCuNi ... Resolution: 1/10 or 1/100°C
- Interface and Counter Modules
3.2 Expansion

3.2.1 RIO

The B&R2003 can be used as **Remote I/O Slave for long distances**, in a 1200m segment, **with 3 repeaters up to 4800m** and provides very fast communication, up to **2MBaud**. The connection is made using RS485 twisted pair lines.

**RIO**

**SLAVE** I/O bus with secure data transfer; max. 8 logic modules

![Module rack + RIO](image1)

**Fig. 4.4: Module rack + RIO**

3.2.2 CAN I/O

The B&R2003 can be used as master and also as slave for **CAN I/O expansions**. The maximum distance extends over **1000m**. The max. transfer speed is **500kBaud**. The connection is made using a **3 conductor CAN cable**.

The CAN bus slave module has its own configuration memory for CAN node parameters. If this memory is not used, the node is started using the **standard configuration**.

**CAN**

**SLAVE** I/O bus with secure data transfer; max. 8 logic modules

![Module rack + CAN I/O](image2)

**Fig. 4.3: Module rack + CAN I/O**

3.2.3 Networks

- CAN with multimaster network
- B&R NET2000 as master/slave network with slave cross traffic
- ETHERNET to connect with higher level systems
3.3 CAN I/O Projects

To work with CAN I/O, a EX470 module must be inserted in the CPU.

![Fig. 4.5: Inserting a CAN slave](image1)

![Fig. 4.6: CAN I/O slave module selection dialog box](image2)
In order for the CAN I/O master to access nodes, the node number on the module has to correspond to the node number in the configuration.

**Fig. 4.7:** CAN node number dialog box

**Fig. 4.8:** Defining DI variables on a CAN slave
Set the “scope” in the pop-up menu so that the relationship between variable and hardware is immediately clear.
Working with CAN I/O (continued)

If the task is compiled and transferred to the controller, the additional drivers are also automatically copied in the “System” area.

![System area with “canio” master](image_url)

**Fig. 4.11: System area with “canio” master**
Example

● Read an analog input from the CAN I/O and output it somewhere else.
● Watch the LEDs to see when the analog values **do not change**
● Watch the LEDs to see when the analog values **change**
● Remove the CAN cable and watch the output

Project Name: can_2003
LAD Name: av_can
Resource: C#3
4 B&R AUTOMATION TARGET 2005

4.1 Main Unit

- The system performance is exceptionally high because of cooperation between the processors on the CPU and parallel processor capability.
- The parallel processors extend the performance palette of this system in various directions, e.g.: very fast I/O in the μs range, interface expansions as well as positioning and CNC applications.
- Expansions are possible using local expansions, remote slaves for longer distances up to 4800m or network and field bus system connections, e.g.: CAN.

4.1.1 Backplane

A backplane can have a maximum of 15 slots.
Slots that are not used should be covered by dummy modules.

4.1.2 Power Supply

Supply voltages in the DC range from 18-30V and in the AC range from 82-264V, 47-63Hz. The power supplies are protected on the primary side by a fuse and on the secondary side by an internal current limiter. The power supply is always on the outer left of the backplane module.

4.1.3 CPU, User Memory and Parallel Processors

The high performance is reached using a processor with integrated RISC. All PCC systems distinguish themselves by their real-time capable, multitasking operating system. SRAM and FPROM are available as application memory. The user is provided a modular IF concept. The interfaces can be switched using software and be operated as either online or data interface.
4.1.4 Configuration Possibilities

Parallel processors are special features of the B&R2005 system which can be used as a CPU if no CPU is inserted.

![Fig. 4.12: Rack with XP152](image)

4.1.5 I/O Modules and Terminals

The B&R2005 system offers an extensive palette of:

- Digital Modules: Relay or transistor version
- Analog Modules:
  - 0-20mA and ±10V
  - PT100 / FeCuNi / NiCrNi
  - Resolution: 1/10 or 1/100 °C, 1/10 F
- Interface, Counter and Network Modules
- Positioning and CNC Modules
4.2 Expansion

The B&R2005 can use both local and remote expansion.

4.2.1 Expansion

The expansion system is used for local expansions. The expansion master is an EX350 in the power supply. The expansion slave is a power supply. With local expansions, there can be a maximum of 2m between the individual stations and up to 4 B&R2005 expansions, max. 52 IO modules can be used. The expansion cable is used as transfer media.

4.2.2 RIO

The remote I/O system consists of a remote I/O master, the system module EX150, and a remote I/O slave, as power supply. Remote networks can have max. 32 stations in a segment and can be used for distances up to 1200m. The transfer rate that can be obtained is 2 MBaud. Using 3 repeaters, a max. of 121 stations can be connected at a distance of 4800m.

RS485 twisted pair is used as transfer media.

4.2.3 CAN I/O

The CAN I/O system consists of at least one master, as CPU or parallel processor with CAN interface, and up to 63 CAN slaves, as B&R2003 with CAN bus slave module. The maximum expansion can have 63 stations. Several masters can be used. Distances up to a max of 1000m can be reached. The max. transfer rate is 500kBaud. A 3 conductor CAN cable is used as transfer media.

4.2.4 Networks

The networks supported include the following networks:

- ETHERNET
- PROFIBUS
- B&R NET2000
- CAN
- Frame Driver
- Connections to systems from other manufacturers
4.3 RIO Projects

In order to work with B&R RIO, a 3EX150 RIO master must be installed on the rack. A RIO slave can be connected there.

![Fig. 4.14: Inserting a RIO slave]

The settings used for the configuration of the RIO stations can be found in the shortcut menu item **RIO Properties**.

![Fig. 4.15: RIO I/O slave module selection dialog box]
Working with B&R RIO (continued)

In order for the RIO master to access the slave nodes, the node number on the module has to correspond to the node number in the configuration.

Fig. 4.16: RIO node number dialog box

Fig. 4.17: Selecting the power supply for the RIO slave module
Working with B&R RIO (continued)

Assigning variable names for inputs and outputs.

Fig. 4.18: Assigning I/O variables

Fig. 4.19: Task with B&R RIO variables
Working with B&R RIO (continued)

If the task is now compiled and transferred to the controller, the additional drivers are also automatically copied in the “System” area and also transferred to the target.

Fig. 4.20: System area with “rio_lib” (Remote I/O Library)
Example

- An output is activated when a positive edge occurs on an input.
- The output is deactivated again when a negative edge occurs on a second input.
- Break the connection to the slave by unplugging the cable. What happens to the slave? What happens to the master?

Project Name: rio_2005
LAD name: rio_pump
Resource: C#2
5 B&R AUTOMATION TARGET 2010

5.1 Main Unit

- The high end system of the B&R2000 series.

- The system performance is a result of cooperation between several processors and the CPU,
  - I/O Processor to read from / write to IO simultaneously
  - RISC to service interfaces simultaneously
  - DPR Controller to mask / negate digital I/O points
  the modular application memory which can be sent in for a project update if a modem service is not available and parallel processor capability.

- Up to 99 modules can be installed on the I/O bus. Each module shows a separate module number on the status display. The terminal blocks are monitored and directly coded to the module which prevents mix-ups when a large number of I/O modules are used.

- The parallel processors extend the performance palette of the system in various directions:
  - IPs, as intelligent peripherals: drum sequencers, injection molding applications
  - PPs, as parallel processors: Multiprocessor, interface expansions, etc.

- The supply principle allows a system with redundant power supplies to be created.

- Expansions are possible using local expansions, remote slaves for longer distances up to 4800m or network and field bus system connections.
5.1.1 Backplane

The system has a modular system and I/O bus.
An I/O bus segment can be up to 20 slots long.
The system bus and the I/O bus both require a terminating resistor.
Slots that are not used should be covered by dummy modules.

5.1.2 Power Supply

Supply voltages in the DC range from 18-30V and in the
AC range from 82-264V, 47-63Hz.
The power supplies are protected on the primary side by a fuse and on the secondary
side by an internal current limiter.
The power supplies are then connected to the I/O bus.
Supply redundancy can be obtained by using twice as many power supplies as
needed.

The 24V secondary voltage can be switched forward to a terminal using the toggle
switch on the AC power supply, PS740.
5.1.3 CPU, User Memory and Parallel Processors

The high performance is reached using **several processors** (main processor, RISC, I/O processor and DPR controller). All PCC systems distinguish themselves by their **real-time capable, multitasking operating system**. The application memory (SRAM and FPROM) is modular and can therefore also be sent in if a modem service is not available.

The CPU is available in different versions. The only differences between the CP100 and the CP104 are the interfaces. Depending on the CPU, an RS232, a modem capable RS232 and RS422/485 interface or an RS232, a modem capable RS232 and CAN interface are available. The CP200 has a much higher calculation performance, 4 to 8 times, and more interfaces, an RS232, a modem capable RS232, a CAN and RS422/485 interface.

The **interfaces** can be **switched using software** and operated as either **online or data interface**.

5.1.4 I/O Modules and Terminals

The B&R2010 system offers an extensive palette of:

**Digital Modules Relay or transistor version**

<table>
<thead>
<tr>
<th>Analog Modules</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-20mA and ±10V</td>
<td>12Bit</td>
</tr>
<tr>
<td>PT100 / FeCuNi / NiCrNi</td>
<td>1/10 or 1/100 °C</td>
</tr>
</tbody>
</table>

**Interface counter modules and network modules**

Multifunction module, drum sequencer,
Positioning module, parallel processors, interfaces
5.2 Expansion

5.2.1 Expansion

The B&R2010 can use both local and remote expansion.

The expansion system is used for local expansions. The expansion master is an I/O module, EX302.

The *expansion slave*, EX301, is the *first module in the expansion station* and must be inserted on a *terminated backplane*, BP202. Local expansions can have a max. of 20 modules and there can be a max. of 2m between the individual stations.

Up to 9 B&R2010 expansions with a max. of 99 modules can be used. The expansion cable is used as transfer media.
5.2.2 RIO

Two different remote systems can be used, Remote I/O and CAN I/O.

The **Remote I/O** system consists of a remote I/O master, the system module EX100, and a remote I/O slave, EX200.

The remote I/O slave is the first module on the remote station and must be installed on a terminated backplane, BP202.

Remote networks can have max. 32 stations in a segment and can be used for distances up to 1200m. The transfer rate that can be obtained is 2 MBaud.

A max. of 121 stations can be connected at distances up to 4800m using a max. of 3 repeaters.

RS485 twisted pair cabling is used as transfer media.
5.2.3 CAN I/O

The **CAN I/O** system consists of at least one master. A CPU or parallel processor with CAN interface, and up to 63 CAN slaves, as B&R2003 with CAN bus slave module. The maximum expansion can have 64 stations. Distances up to a max of 1000m can be reached. The max. transfer rate is 500kBaup.

A 3 conductor CAN cable is used as transfer media.

5.2.4 Networks

The following networks are also supported for data transfer:

- ETHERNET
- PROFIBUS
- CAN
- B&R NET2000
- Frame Driver
- Connections to systems from other manufacturers
6 B&R AUTOMATION TARGET LOGIC SCANNER

6.1 Main Unit

- Combination of control and visualization tasks compact in a single device.
- IPC5xxx handles visualization tasks.
- LS251 handles control tasks and is completely independent of the IPC5xxx operating system.
- High speed data exchange between LS251 and IPC5xxx via the PCI bus.
- Expansion possibilities exist for IO using RIO or CAN to connect to a network.

6.1.1 PCI Bus

The PCI bus is connecting element between the LS251 and IPC5xxx. It is used as a high speed data exchange media.

6.1.2 Power Supply

The LS251 uses either the IPC5xxx power supply directly or an external supply (independent of the IPC5xxx supply) provided by the LS079 expansion card.

6.1.3 CPU, Application Memory

The high performance is reached using a processor with integrated RISC. All PCC systems distinguish themselves by their real-time capable, multitasking operating system. SRAM and FPROM are available as application memory. The user is provided a modular IF concept, with the LS071. The interfaces can be switched using software and operated as either online or data interface.
6.2 Expansion

The LS251 can be connected to I/O using RIO or CAN

6.2.1 RIO

The LS251 has a RIO interface onboard for a connection to IO modules from the B&R 2003/2005/2010 families.

6.2.2 CAN IO

The LS251 has a CAN interface onboard for a connection to IO modules from the B&R 2003 family.
7 B&R AUTOMATION TARGET IPC2XXX

7.1 Main Unit

- Increased processor performance with support of an FPU
- Large supply of memory with possibility for file management
- Compact design with flexible network connections

7.1.1 Power Supply

Integrated DC power supply with a supply voltage of 24V.

7.1.2 CPU, Application Memory

The IPC2xxx, having an Intel 486/DX5 with FPU and memory in the megabyte range, provides high performance. The user now also has PC resources, such as floppy disk and hard disk with a file management system for effective development of applications. The interfaces can be switched using software and operated as either online or data interface.

7.1.3 AutomationRuntime

To use the **IPC2xxx** as AutomationTarget, **AutomationRuntime AR102** is installed as real-time operating system from a set of diskettes. These installation diskettes are created by the AutomationSoftware install kit.
7.2 Expansion

7.2.1 ISA I/O

With the LS301.4 card, 16 DI, 16 DO, 4AI, 2AO can be connected directly to the IPC2xxx via the ISA bus.

7.2.2 CAN IO

With the LS172.4 card, B&R CAN IO can be connected to the IPC2xxx via the ISA bus. The LS172.4 has two CAN interfaces.

7.2.3 Networks

- CAN
- Ethernet
- Serial
8 B&R AUTOMATION TARGET IPC5XXX

8.1 Main Unit

- Increased processor performance with support of an FPU.
- Large supply of memory with possibility for file management.
- Compact and modular design with flexible network connections.

8.1.1 Power Supply

Integrated AC or DC power supply. 24V DC supply voltage and 100-240V AC supply voltage, 47-63Hz.

8.1.2 CPU, Application Memory

The IPC5xxx, having a Pentium processor with FPU and memory in the megabyte range, provides the highest performance. The user now also has PC resources, such as floppy disk and hard disk with a file management system for effective development of applications. The interfaces can be switched using software and operated as either online or data interface.

8.1.3 AutomationRuntime

The following AutomationRuntimes are available for use with IPC5xxx as AutomationTarget:

- **AR105 - embedded.** The AT only executes control tasks.
- **AR010 - NT.** The AT executes high priority and deterministic control tasks. NT is handled in the idle time which makes it possible to run the visualization on the same target.

AutomationRuntime is installed as real-time operating system from a set of diskettes. These installation diskettes are created automatically by the AutomationSoftware install kit.
8.2 Expansion

8.2.1 ISA I/O

With the LS301.4 card, 16 DI, 16 DO, 4AI, 2AO can be connected directly to the IPC5xxx via the ISA bus.

8.2.2 Expansion

With the LS191 card, B&R 2005/2010 IO can be connected to the IPC5xxx as local expansion via the PCI bus.

8.2.3 CAN IO

With the LS172.4 card, B&R CAN IO can be connected to the IPC5xxx via the ISA bus. The same possibilities are offered by the LS172.6 using a faster PCI bus connection.

8.2.4 Networks

- CAN
- Ethernet
- Serial
9 B&R2000 OVERVIEW

Remote Master Network  Expansion Master

Remote Slave Expansion Slave

Power Supply with Expansion Slave

Up to 31 remote slave stations with repeater 121

I/O BUS

RIO Slave Module

Up to 4 expansion stations

Can-Bus Slavemodul

Up to 63 Server Stationen

Network

IPC + AR

CAN

PROVIT 5500
B&R AUTOMATION NET

1 OVERVIEW .................................................................................................................. 2

2 B&R AUTOMATION NET .......................................................................................... 3

3 COMMUNICATION PRINCIPLES ............................................................................. 4

4 ACCESS TO B&R AUTOMATION NET ................................................................. 5
    4.1 B&R Automation Net - PVI ................................................................. 5
    4.2 B&R Automation Net - Routing .......................................................... 6
    4.3 B&R Automation Net - INA Client FBKs ......................................... 9
1 OVERVIEW

B&R Automation Net
   Overview of the communication model for B&R Automation Net and its components.

Communication Principles

Access to B&R Automation Net
   The user can access Automation Net for visualization and for the programming device via PVI. From the application, access takes place using INA Client FBKs. Routing possibilities are offered which allow new solution methods for simple handling of network tasks.
B&R Automation Net (AN) allows communication between B&R Automation Runtime, B&R Automation Target and also other stations on the network. In general, B&R AN represents a cloud of communication between B&R and other components.
3 COMMUNICATION PRINCIPLES

B&R Automation Net makes it possible for every communication station to exchange and edit all types of program objects and/or process variable objects.

B&R Automation Net is:

- independent of the operating system used
  B&R Automation Runtime, Windows 95/98/NT/2000, etc.
- independent of the media used
  RS232, CAN, Ethernet, Profibus, Modem, Memory, etc.
- independent of the transfer protocol used
  INA2000, NET2000, Mininet, etc.
4 ACCESS TO B&R AUTOMATION NET

Communication within AN can be transparent, but different interfaces must be used at the end points where the information is actually processed on the respective operating system.

Therefore, the corresponding access method is required for each operating system.

4.1 B&R Automation Net - PVI

The Process Visualization Interface (PVI) is a component of AN and establishes the connection to the B&R Industrial PC’s environment as common interface for all Windows based programs.
4.2 B&R Automation Net - Routing

Routing generally refers to a connection between several PCCs for programming. Its main task is forwarding data addressed on another PCC.

The interface settings are made in AS using menu item Extras: Options

Fig. 5.1: Connection settings
4.2.1 Online Configuration

Select the online configuration for communication to the local PCC. Automation Studio provides the following preprogrammed configurations:

- Serial = RS232
- CAN
- SHARED = LS251

Additionally, the user can save other, specific configurations under different names.

4.2.2 Interface

Here, parameters for the selected online interface can be defined in a dialog box. The parameters can also be entered as text using “Extra Settings”.

4.2.3 Connection Parameters

Additional settings for the device parameters when using CAN, Profibus and TCP/IP Ethernet.

e.g. node number for CAN or TCP/IP Ethernet

4.2.4 Target System Path

Settings for routing between controllers

  e.g.: “/CN=CAN.3”

This setting means that a CAN connection will be created from the local controller to the next controller. The ending “.3” is the set node number of the next station.

4.2.4 Activate Remote Connection

The user can make settings for a remote connection between PCs via TCP/IP.

Note

More than one routing path can be entered !!
Example

Create the network shown below with the help of your trainer and establish communication between the controllers and Automation Studio.

- Create the network
- Program a tasks with a toggle output
- Transfer and test the program

Project Name: rout_can
Task Name: toggle
Resource: ?
4.3 B&R Automation Net - INA Client FBKs

Today, many automation solutions use distributed intelligence. Tasks are distributed over several automation targets. The connection between the automation targets is made using Automation Net.

The communication services, such as reading or writing PVs, are already provided to the programmer by Automation Runtime. This communication provided by Automation Runtime is called the INA Server.

Access of the communication services by the application takes place using function blocks, INA Client FBKs.

With INA Client FBKs, PVs can now be read from or written to by another automation target using their names via Automation Net. This makes exchanging data between B&R controllers much easier and more transparent for the user.

Advantages of the INA Services:

- Connections are made by setting parameters and no longer need to be programmed
- Complete performance of the INA2000 protocol
- Change the physics simply by changing the parameters
- Fast and simple use of network technologies without having to make extensive changes to source code
PROJECT GUIDELINES I

1 OVERVIEW ........................................................................................................... 2

2 PROJECT CREATION ......................................................................................... 3

3 PROGRAMMING CONVENTIONS ................................................................. 4
   3.1 Identifier ............................................................................................... 4
   3.2 Data Types ............................................................................................ 4
   3.3 Directory Structure ............................................................................ 4
   3.4 Software Module Names .................................................................... 5
   3.5 Task Names .......................................................................................... 5
   3.6 Data Module Names ........................................................................... 6
   3.7 Variable Names .................................................................................... 7
   3.8 Constant Names .................................................................................... 10
   3.9 Alias Process Variables for Function Blocks .................................. 11
   3.10 Assigning Revision Numbers ............................................................. 11
1 OVERVIEW

For larger projects, it is important that uniform guidelines are followed during development and programming.

This should make it easier for several programmers to work together on a project.

This should also make programs more clear for the programmer and for others. The ability to expand and maintain the programs is improved considerably.

Project Creation

Short overview of the process of creating projects. This information also depends on the branch and the respective projects, but can be used as a basis and adapted for the project being created.

If guidelines already exist for projects, they should be followed!

Programming Conventions

This concerns assigning names to tasks and also variables. Variable names and constant names are designed for 32 characters. These characters should be used in as meaningful a manner as possible. Task names are limited to 8 characters. The nesting depth for structures is limited to 16.

Notes on Literature

This chapter is a excerpt from the B&R application programming guidelines which is described in detail in the appendix.

Further Literature:

Code Complete : A Practical Handbook of Software Construction
by Steve C McConnell

Paperback - 857 Pages (May 1993)

Microsoft Press;
ISBN: 1556154844
2 PROJECT CREATION

Details concerning the individual points can be found in Project Guidelines II

- Internal preliminary discussions
- Creating a project
- Project discussions

- Software conception phase
  Defining or adjusting the programming conventions

- Coding
- Testing
- Startup
- Documentation
- Archiving

Note
In this chapter, only the software conception phase will be looked at in detail.
3 PROGRAMMING CONVENTIONS

3.1 Identifier

AS is a programming tool that makes it possible to assign meaningful variable names.

- Variable name: 32 characters
- Constant name: 32 characters
- Task names: 8 characters
- File names: W95/W98/WNT naming conventions
- Nesting depth for structures: 16 levels

Programming conventions for B&R AutomationStudio are described in the following sections.

They should be followed when creating software, during startup and also for later software changes.

3.2 Data Types

In all programming languages supported by AutomationStudio, the IEC Data Types should be used. Also ANSI C !!!

This simplifies changing to other architectures considerably because only the platform specific sections have to be rewritten.

3.3 Directory Structure

As standard, AutomationStudio saves all source files in the CPU path for the project.

With the programming language ANSI C, it is possible to create subdirectories for source and include files which makes them much clearer.

.. //Cpu//SRC
.. //Cpu//INC
3.4 Software Module Names

3.4.1 General Information

During the software conception phase, the application is divided into smaller and smaller units which provide the required functionality.

This type of software module should be a closed unit. It handles a certain task and has a precisely defined interface to other software modules.

A software module can consist of one or more tasks and data modules.

The module concept should also be used for function blocks and functions.

3.4.2 Guidelines for Module Names

A module is defined using a combination of three characters. This character combination can consist of letters and/or numbers.

The character combination is found in all elements that belong to this module such as tasks, data modules, global variables, function block names and function names.

3.5 Task Names

It should be clear that a task belongs to a software module from its name. Task names are presently limited to 8 characters and are defined as follows:

<table>
<thead>
<tr>
<th>Abbreviation:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm_</td>
<td>Code for the module</td>
</tr>
<tr>
<td>ttttt</td>
<td>Code for the task in the module</td>
</tr>
</tbody>
</table>

Tab. 6.1: mm_ttttt

To improve clarity, module and task identification codes should be separated by an underline.

Example: mi_drv ... machine interface, driver
3.6 Data Module Names

It should be clear that a data module belongs to a software module from its name.

In certain cases, it can be necessary to enter the memory type for the data module in the name. For example: Backup copy in Flashprom, if the data module is written to by the application.

Data module names are presently limited to 8 characters and are defined as follows:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm_</td>
<td>Code for the module</td>
</tr>
<tr>
<td>s</td>
<td>Memory type</td>
</tr>
<tr>
<td>tttt</td>
<td>Code for the data module</td>
</tr>
</tbody>
</table>

Tab. 6.2: mm_stttt

<table>
<thead>
<tr>
<th>Prefix s</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>No entry</td>
</tr>
<tr>
<td>p</td>
<td>Eprom/Flashprom</td>
</tr>
<tr>
<td>x</td>
<td>Fix Ram</td>
</tr>
<tr>
<td>r</td>
<td>Ram</td>
</tr>
</tbody>
</table>

Tab. 6.3: Memory type

Example

**mi_para**
Data module in the MI module with the name Para, the memory type is not entered.

**mi_xpara**
Data module in the MI module with the name Para, which is in FIX RAM.
3.7 Variable Names

AutomationStudio differentiates between variables with the following scope:

- global variables
- local variables
- C variables (see chapter ANSI C)

Variable names can have up to 32 significant characters.

Variable should always be defined with the smallest possible scope. When using PCC global variables, very complex structures can easily be created which makes the application logic difficult to understand. Additionally, PCC global variables make it considerably more difficult to reuse and maintain codes.

As the size of the project increases, it becomes more important to be able to quickly recognize the scope of variables. This can be made easier by using a type prefix with variable names.

3.7.1 Global Variables

Applications for global variables:

- Communication between Tasks
- IO Connection to Hardware

Global variables are to be defined with “_GLOBAL” in ANSI C.
The separation of the sections can be shown using underlines or capitalization.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Type prefix for scope of the variable</td>
</tr>
<tr>
<td>tt</td>
<td>Type of variable</td>
</tr>
<tr>
<td>MMM</td>
<td>Code for the module (only capitals and/or numbers)</td>
</tr>
<tr>
<td>nnnnnnnnnn..</td>
<td>Variable name consists of letters, numbers and underline. The name must begin with a lower case letter or an underline.</td>
</tr>
</tbody>
</table>

Tab. 6.4: Names of Global Variables

<table>
<thead>
<tr>
<th>Type Prefix p</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>PCC global variable</td>
</tr>
<tr>
<td>None</td>
<td>C local variable and local variables in IEC languages</td>
</tr>
</tbody>
</table>

Tab. 6.5: Description of the type prefix p

<table>
<thead>
<tr>
<th>Type tt:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Di</td>
<td>Digital input (BOOL)</td>
</tr>
<tr>
<td>Do</td>
<td>Digital output (BOOL)</td>
</tr>
<tr>
<td>Ai</td>
<td>Analog input (INT)</td>
</tr>
<tr>
<td>Ao</td>
<td>Analog output (INT)</td>
</tr>
<tr>
<td>P_</td>
<td>Pointer (only meaningful in &quot;C&quot;)</td>
</tr>
</tbody>
</table>

Tab. 6.6: Type description of variables

Example

IO Inputs/Outputs:

gDiHTGStart, gDoHTGStop, gAiHTGzyTemp, gAoHTGzyClock, etc.
3.7.2 Access of Hardware IO

The access of hardware data points can occur in each software module exactly once.

An exception to this is LAD programming. Here, inputs can be accessed more often. Outputs can only be written to once.

For all other programming languages, the following is valid:

- At the beginning of the module, the hardware IOs in the hardware data point buffer are copied to the local module status structure.
- At the end of the module, the buffer is written to the outputs.
- A substructure HwIO in the local module status structure is used as buffer.

This also has the advantage that changing the hardware data points between normally open to normally closed contacts only affects one location in the program.

IMPORTANT

Outputs are only allowed to be written to by a module one time!

3.7.3 Local Variables

- Names for local variables can contain all number and letter combinations.
- Exchanging local data and commands can take place using the respective prefixes “C_” and “S_”.
- Local variables are to be defined with “_LOCAL” in ANSI C. This variable type can be viewed with “Watch”.

Additionally, they have to begin with the prefix „I“ in ANSI C, the prefix is not needed in all other IEC languages.

Example in IEC languages:

resAverage, intCounter, ...
3.7.4 Initializing Variables

Variables are principally only to be initialized in the “INIT SP” for the main task or module.

The variable initialization in the variable declaration window is always to be set to “remanent”.

3.8 Constant Names

- Constant names can have up to 32 significant characters.
- Self-defined constants are always written in capitals.

Constants defined by B&R standard software, e.g. NC software are exceptions. They are not allowed to be changed during the creation of the application!

3.8.1 Constant Definition in ANSI C

In ANSI C, there are three possibilities to define constants.

- Constants that are valid throughout the system
  These are always written in capitals and are not allowed to have underlines.

  Example: TRUE, FALSE, UNDEF, ...

- Constants with software specific validity
  Constants are principally always valid throughout the system, but the use of a specific code allows them to be assigned to a program.

- Commands
  Constants belonging to a spezial module are labeled with the module code and an underline.

  Example: MI_START, MI_STOP, C_START, C_STOP
3.8.2 Reserved Constants

These constants are written in capitals but are labeled mit “nc” in lower case at the beginning.

Example: ncON, ncOFF, ncINIT

These constants are not allowed to be changed by the user.

3.9 Alias Process Variables for Function Blocks

In alias names of function blocks, the function block name always has to be first. The description is separated by an underline.

Example: TON_valve, TOF_motor

3.10 Assigning Revision Numbers

In all headers where version numbers are assigned, the following procedure should be used:

The version number consists of 4 characters.

**xx.yy:**

*yy ..* increased by 1 with each change
*xx ..* increased by 1 for major changes. *yy* becomes 0.

Version numbers are assigned in hex format.

IMPORTANT

Detailed literature concerning this topic is available.
1 OVERVIEW

B&R offers the right programming language for every application and for every programmers preference. This includes:

- Ladder Diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD  Contact, logic and function operations are combined here in a single user interface. Ladder diagram is the simplest form of programming digital and analog processes because of its similarity to a circuit diagram.

IL  Instruction list is similar to a machine language. It can be used like LAD for logic operations.

ST  This high level language is a clear and powerful programming language for automation systems. Simple standard constructs guarantee fast and efficient programming.

SFC  A sequential language that was developed to separate a task into clear units. Sequential Function Chart (SFC) is well suited for processes where states change in steps, for example: automatic carwash

AB  This B&R high level language is a clear and powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient programming. Previously PL2000

ANSI C  This high level language is a powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient application programming.
2 SFC SYNTAX

In this chapter, we will have a detailed look at the language Sequential Function Chart (SFC).

SFC is a **graphically structured language** which eases sequential control.

This IEC61131-3 language is based on GRAFCET, an important French programming language.

The SFC programming symbols are divided into the following groups:

- Initialization step
- Steps – sequential states
- Actions – for IEC steps
- Defining characters for actions
- Branches
- Transitions
  - Transfer conditions between steps
2.1 Steps

2.1.1 Init Step

Every SFC program contains a special step. The initialization step. This INIT step is symbolized by a double rectangle. The sequence begins with this step each time the controller is started.

2.1.2 Normal Steps

„Normal“ steps are symbolized by a rectangle. The rectangle contains the name of the respective step.

A variable exist for each step with the name of the step. The status of the step can be read from this variable.

For simplified steps, this variable has data type “BOOL”. For IEC steps, this is a structure variable.
2.2 Actions

- Actions contain the actual program code
- Actions are always assigned to a step

They are several types of actions:
The languages IL, ST, LAD, AB and SFC can be selected.

2.2.1 Simple actions

The simple form of actions are called “Action”. The code contained in them is always executed when the action is active. You can see that an action is assigned to a step because a small triangle on the top right of the symbol appears.

2.2.2 Entry Action

The code for the entry action is executed once when the step becomes active. That means whenever the step status changes from inactive to active. If a step contains an entry action, a small square with the letter “E” is shown in the bottom left corner of the step symbol.

2.2.3 Exit Action

The code is executed once when the action status changes from active to inactive. The exit action has an “X” in the lower right corner.

Steps can contain both “entry actions”, and “exit actions”!
2.3 Transitions

Transitions are symbolized by “” symbols at the connection point between the steps. The transition conditions are on the right next to the symbol.

The transition conditions can be a simple BOOL variable or logically linked variables where the result must be TRUE or FALSE!

The space next to the transition is limited, therefore transitions can be entered in a separate editor. Only one link can be entered in this editor. Command sequences are not allowed. If the transition is the result of a logical link, this is shown by a small triangle.

Languages IL, ST, LAD and Automation Basic can be selected.

2.4 Jumps

Jumps are used to create loops and repeats. Jumps are represented by a jump symbol. The name of the jump target is shown under the jump symbol.
2.5 Branches

2.5.1 Alternative Branch
Using alternative branches, it is possible to continue processing in one of several branches depending on the transition condition. If several transition are TRUE, then the left branch is processed. To make things clearer, transitions should be selected so that only one is active at all times.

![Alternative Branch Diagram]

2.5.2 Parallel Branch
Using parallel branches, it is possible to process several branches in “parallel”. The steps in the respective branches are naturally processed one after the other. The order moves from left to right. The transition which where the branches come together is evaluated when the last step is active in each branch.

![Parallel Branch Diagram]
Example

Use the editor to create a sequence with normal steps, transitions, alternative branches and parallel branches to get used to the editor.

Add steps and transitions.

Delete steps and transitions, etc.

IMPORTANT

To select steps and transitions, you must work with the <SHIFT>+<Cursor> keys.

Project Name: sfc_pro1
Task Name: sfc_edit
Resource: C#2

Note

Variables are requested directly by the editor and should be defined as local.
2.6 IEC Steps

2.6.1 IEC Actions

IEC actions can only be connected to IEC steps. They are recognizable by the horizontal line on the right corner.

The actions are represented by “banners” which are connected to the horizontal line.

The “banner” is divided into 2 columns. The first column contains a letter that specifies the execution rate. Some types of execution require a time entry which is entered next to the letter. Constants, variables and literals of type “TIME” are valid as time entries. The second column shows the name of the action being executed.

Several IEC actions can be connected to IEC steps. For example, an IEC action can be continually activated in a step using execution type “S” and then deactivated in another step with “R”.

“Entry actions” and “exit actions” are also possible in IEC steps.
2.6.2 Execution Types for IEC Actions

The following execution types are possible for IEC actions:

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
</table>
| N         | Not stored  
Action is executed as long as the step is active |
| R         | Reset  
Action is not executed. Stored status of the action is deleted. |
| S         | Set  
Action executed until reset with R. |
| L t#Time  | Limited  
Action executed as long as step is active and the set time has not passed (according to IEC format). |
| D t#Time  | Delayed  
Action executed if the set time has passed (according to IEC format) and the step is still active. |
| P         | Pulse  
Action executed once when the steps becomes active. |
| SD t#Time | Stored and Delayed  
Action becomes active after set time, independent of if the step is still active. |
| DS t#Time | Delayed and Stored  
Action becomes active if the step was active for at least the set time, status is stored after the step has ended. |
| SL t#Time | Stored and Limited  
The action remains active for exactly the set time, independent of if the step remains active. |

Tab. 7.1: IEC actions

2.6.3 Executing IEC Actions

Each IEC action has a structure variable with the name of the action. Element .x of the structure can be used to read the activation status of the action.

Evaluating this value <action name>.x can be done to determine, if an action is active or inactive.

IEC actions are executed as long as they are active. After changing from active to inactive, the actions executed one more time.

During the last execution of the action, the value of <action name>.x = FALSE.
3 PLANNING WITH SFC

Example

Application:

Press start...
Open water valve

Water OK reached...
Close water valve, Switch on stirring mech., Open color valve

Sensor Full reached...
Close color/dispersion valve, Wait 30 sec. Open drain valve Switch on drain pump

Sensor Low reached...
Switch off pump, switch off stirring mech. Close drain valve

Procedure:

- Determine the necessary steps
- Determine the transition conditions, transitions
- Define further programming conventions if necessary, for variable names, etc.
- Create SFC task
- Program steps
- Program transitions
- Test steps
- Define outputs
- Enter code in the steps and test the function
3.1 Planning on Paper
## 3.2 SFC Tools

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Search" /></td>
<td>Search</td>
<td><img src="image" alt="Search forwards" /></td>
<td>Search backwards</td>
</tr>
<tr>
<td></td>
<td>&lt;ctrl&gt;&lt;f&gt;</td>
<td></td>
<td>&lt;shift&gt;&lt;f3&gt;</td>
</tr>
<tr>
<td><img src="image" alt="Prev. step transition" /></td>
<td>Prev. step transition</td>
<td><img src="image" alt="Next step transition" /></td>
<td>Next step transition</td>
</tr>
<tr>
<td></td>
<td>&lt;ctrl&gt;&lt;t&gt;</td>
<td></td>
<td>&lt;ctrl&gt;&lt;e&gt;</td>
</tr>
<tr>
<td><img src="image" alt="Parallel branch right" /></td>
<td>Parallel branch right</td>
<td><img src="image" alt="Parallel branch left" /></td>
<td>Parallel branch left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Use IEC steps" /></td>
<td>Use IEC steps</td>
<td><img src="image" alt="Add action" /></td>
<td>Add action</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Delete action" /></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Select actions" /></td>
<td>Select actions</td>
<td><img src="image" alt="Edit action" /></td>
<td>Edit action</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Link action" /></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Start object" /></td>
<td>Start object</td>
<td><img src="image" alt="Stop object" /></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Break/delete action/step/transition" /></td>
<td>Break/delete action/step/transition</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Tab. 7.2: SFC Toolbar*
3.3 SFC Application

Mixer in SFC.

The steps should always be defined first.

![Fig. 7.2: Defining the steps in SFC](image)

Then the transition conditions are defined.

![Fig. 7.3: Steps with transitions](image)
Mixer in SFC (continued)

When the inputs have been defined, the SFC program can be tested in its basic form.

**Fig. 7.4:** Defining the hardware assignments

**Fig. 7.5:** Testing the steps and the transition conditions
Mixer in SFC (continued)

Double-clicking on the step symbol opens a dialog box that can be used to define the source type for the block.

![Fig. 7.6: Defining the code type](image)

The outputs are defined in the block according to the plan made previously.

![Fig. 7.7: Code for step st_WAIT](image)
Mixer in SFC (continued)

Comments can be added to the individual steps. The local menu can be opened with the right mouse button.

Fig. 7.8: Enter comments for step

Fig. 7.9: "Step Attribute" dialog box
Mixer in SFC (continued)

Code for step “st_WATER”

```
0001 (* Set in Water *)
0002 (* Create Message for Watch ! *)
0003 strMessage := "Set in Water !";
0004 doValveWater := 1
```

Fig. 7.10: Defining the outputs in the step

If an output is set in a step, it can be reset when exiting the step if required by the application.

Fig. 7.11: Defining an “exit action”
Mixer in SFC (continued)

Source code for the exit action.

```
00001 (* EXIT Action for Step Set in Water *)
00002
00003 (* The EXIT Action will be executed only once *)
00004 (* at leaving the step *)
00005 (* The EXIT Action is also used to reset the *)
00006 (* Timer: FUDs like TON to zero again *)
00007
00008 gDoValveWater := 0;
00009
00110
00111
00112
00113
00114
00115
00116
00117
00118
```

Fig. 7.12: Code for the exit action

![SFC Diagram](image-url)

Fig. 7.13: Indication of the exit action
Mixer in SFC (continued)

If we check the settings for the example again, we will see that the sensor inputs are wired as normally closed contacts. That means they must be inverted. For the transitions, it is possible to create this type of simple connection directly. The result of this type of connection must be TRUE or FALSE. Other connections are not allowed.

Fig. 7.14: Defining a transition connection

Fig. 7.15: Code and indicator for a transition link
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERVIEW</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SYNTAX</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Command Groups</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Operator Priorities</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Logical Links</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Arithmetic Operations</td>
<td>6</td>
</tr>
<tr>
<td>2.5</td>
<td>Data Type Conversion</td>
<td>6</td>
</tr>
<tr>
<td>2.6</td>
<td>Logical Comparison Expressions</td>
<td>7</td>
</tr>
<tr>
<td>2.7</td>
<td>Decisions</td>
<td>7</td>
</tr>
<tr>
<td>2.8</td>
<td>Case Statements</td>
<td>8</td>
</tr>
<tr>
<td>2.9</td>
<td>Loops</td>
<td>11</td>
</tr>
<tr>
<td>2.10</td>
<td>Select Statements</td>
<td>13</td>
</tr>
<tr>
<td>2.11</td>
<td>Working with Function Blocks</td>
<td>16</td>
</tr>
</tbody>
</table>
1 OVERVIEW

B&R offers the right programming language for every application and for every programmers preference. This includes:

- Ladder Diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD  Contact, logic and function operations are combined here in a single user interface. Ladder diagram is the simplest form of programming digital and analog processes because of its similarity to a circuit diagram.

IL  Instruction list is similar to a machine language. It can be used like LAD for logic operations.

ST  This high level language is a clear and powerful programming language for automation systems. Simple standard constructs guarantee fast and efficient programming.

SFC  A sequential language that was developed to separate a task into clear units. SFC is well suited for processes where states change in steps, for example: automatic carwash.

AB  This B&R high level language is a clear and powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient programming. Previously PL2000

ANSI C  This high level language is a powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient application programming.
2 SYNTAX

2.1 Command Groups

- Logical Links
- Arithmetic Operations
- Logical Comparison Expressions
- Decisions
- Loops
- Select Statements
- Function Blocks
2.2 Operator Priorities

The use of several operators in a line brings up the question of priority.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brackets</td>
<td>()</td>
<td>highest</td>
</tr>
<tr>
<td>Function Evaluation</td>
<td>Qualifier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Argument List</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>EXP(IN1,IN2)</td>
<td></td>
</tr>
<tr>
<td>Negation</td>
<td>NOT</td>
<td></td>
</tr>
<tr>
<td>Complement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>Modulo</td>
<td>MOD</td>
<td></td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Comparison</td>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
<td></td>
</tr>
<tr>
<td>Equal</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Unequal</td>
<td>&lt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Boolean AND</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>Boolean Exclusive OR</td>
<td>XOR</td>
<td></td>
</tr>
<tr>
<td>Boolean OR</td>
<td>OR</td>
<td>lowest</td>
</tr>
</tbody>
</table>

Tab. 8.1: Operator priorities
2.3 Logical Links

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Logical Links</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>Unary Negation</td>
<td>( a := \text{NOT} \ b; )</td>
</tr>
<tr>
<td>AND</td>
<td>Logical UND</td>
<td>( a := \ b \ \text{AND} \ c; )</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR</td>
<td>( a := \ b \ \text{OR} \ c; )</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive OR</td>
<td>( a := \ b \ \text{XOR} \ c; )</td>
</tr>
</tbody>
</table>

Tab. 8.2: Logical links

Motor\_1 := (In\_1 \ \text{AND} \ (\text{NOT} \ In\_2 \ \text{OR} \ In\_3) ) \ \text{OR} \ In\_4;
2.4 Arithmetic Operations

A decisive factor when deciding to use a high level language is the simplicity when handling arithmetic operations.

Automation Basic provides the basic arithmetic functions for your application such as:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Arithmetic Operations</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>:=</td>
<td>Assignment</td>
<td>a := b;</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>a := b + c;</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>a := b - c;</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>a := b * c;</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>a := b / c;</td>
</tr>
<tr>
<td>mod</td>
<td>Whole number division</td>
<td>a := b mod c;</td>
</tr>
</tbody>
</table>

Tab. 8.3: Arithmetic operators

2.5 Data Type Conversion

2.5.1 Implicit Data Type Conversion

If different data types come together in operations, the compiler carries out an implicit data type conversion.

<table>
<thead>
<tr>
<th>Data type</th>
<th>BOOL</th>
<th>SINT</th>
<th>INT</th>
<th>DINT</th>
<th>USINT</th>
<th>UINT</th>
<th>UDINT</th>
<th>REAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SINT</td>
<td></td>
<td>x</td>
<td>INT</td>
<td>DINT</td>
<td>USINT</td>
<td>UINT</td>
<td>UDINT</td>
<td>REAL</td>
</tr>
<tr>
<td>INT</td>
<td></td>
<td>x</td>
<td></td>
<td>DINT</td>
<td>INT</td>
<td>UINT</td>
<td>UDINT</td>
<td>REAL</td>
</tr>
<tr>
<td>DINT</td>
<td></td>
<td>x</td>
<td>DINT</td>
<td></td>
<td>DINT</td>
<td>INT</td>
<td>UDINT</td>
<td>REAL</td>
</tr>
<tr>
<td>USINT</td>
<td></td>
<td>x</td>
<td>USINT</td>
<td>INT</td>
<td>DINT</td>
<td>UINT</td>
<td>UDINT</td>
<td>REAL</td>
</tr>
<tr>
<td>UINT</td>
<td></td>
<td>x</td>
<td>UINT</td>
<td>DINT</td>
<td>UINT</td>
<td>UDINT</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>UDINT</td>
<td></td>
<td>x</td>
<td>UDINT</td>
<td>UDINT</td>
<td>UDINT</td>
<td>UDINT</td>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td>x</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
<td>REAL</td>
</tr>
</tbody>
</table>

Tab. 8.4: Implicit data type conversion

2.5.2 Explicit Data Type Conversion

Explicit data type conversion takes place using functions from the STANDARD library.

e.g.: DINT(bool_var)
2.6 Logical Comparison Expressions

High level languages such as Automation Basic allow branches to be easily create using comparison operations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Logical Link Expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal</td>
<td>IF a = b THEN ...</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Unequal</td>
<td>IF a &lt;&gt; b THEN ...</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>IF a &gt; b THEN ...</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
<td>IF a &gt;= b THEN ...</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>IF a &lt; b THEN ...</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
<td>IF a &lt;= b THEN ...</td>
</tr>
</tbody>
</table>

Tab. 8.5: Logical comparison expressions

2.7 Decisions

<table>
<thead>
<tr>
<th>Decision</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>a := b * c;</td>
<td>Calculations</td>
</tr>
<tr>
<td>THEN</td>
<td>IF a &gt; 0 THEN</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>result := 1;</td>
<td>1st Condition</td>
</tr>
<tr>
<td>ELSE IF</td>
<td>ELSE IF a = 0 THEN</td>
<td>Condition met</td>
</tr>
<tr>
<td></td>
<td>value := 100;</td>
<td>2nd Condition</td>
</tr>
<tr>
<td>ELSE</td>
<td>ELSE</td>
<td>Otherwise ...</td>
</tr>
<tr>
<td></td>
<td>result := 0;</td>
<td>No condition met</td>
</tr>
<tr>
<td>ENDIF</td>
<td>ENDIF</td>
<td>End of decision</td>
</tr>
</tbody>
</table>

Tab. 8.6: IF statements / decisions

- First condition true: Execute the THEN branch
- Second condition true: Execute the ELSE IF THEN branch
- Conditions are false: Execute the ELSE branch

The ELSE IF BRANCH and the ELSE branch of an IF statement are optional.
2.8 Case Statements

CASE statements all fast access of various actions depending on the value of a variable. We will clarify the use of CASE statements using a display switch with a selection dial that has several positions:

<table>
<thead>
<tr>
<th>Case Statement</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE OF ACTION 1:</td>
<td>ACTION 1: Display := OVERVIEW;</td>
<td>Only for one position</td>
</tr>
<tr>
<td>ACTION 2,5:</td>
<td>ACTION 2: ACTION 5: Display := NOT_SUPPORTED;</td>
<td>Valid for position 2 or 5</td>
</tr>
<tr>
<td>ACTION 6..10:</td>
<td>ACTION 6..10: Display := SET_VALUE;</td>
<td>Valid for positions 6 to 10</td>
</tr>
<tr>
<td>ACTION 11..20:</td>
<td>ACTION 11..20: Display := ACT_VALUE;</td>
<td>Valid for positions 11 to 20</td>
</tr>
<tr>
<td>ELSEACTION</td>
<td>ELSEACTION: Display := ERROR;</td>
<td>All other positions</td>
</tr>
<tr>
<td>ENDCASE</td>
<td>ENDCASE</td>
<td>End of the CASE statement</td>
</tr>
</tbody>
</table>

The **expression**, between CASE and OF must be a UINT data type and can have a value between 0 and 65535!

Only **whole number values** can be used as steps.

A **colon** must be placed after the step number.

Several numbers can follow the same statement block:

- Fields with **progressive numbers** e.g.: 6..10: can only have 2 points between the numbers!
- **Lines in a list** of number lines must be directly below the previous line e.g.: ACTION 2:
  
  ACTION 5:

The ELSEACTION STATEMENT BLOCK PROCESSES ALL NUMBERS THAT ARE NOT LISTED. ONLY **ONE STEP PER CYCLE** is executed.
Example

The CASE statement is often used when making selections. In this example, we want to use the CASE statement for simple elevator control.

The elevator control should be created for a building with 3 floors. The following steps must be used:

- Definition of the constants for the ACTIONS

  BASEMENT   UINT      0  
  FIRST FLOOR UINT      1  
  SECOND FLOOR UINT      2  

- Initialization of the floor positions

  basement_pos = 0  
  firstfloor_pos = 500  
  secondfloor_pos = 1000  

<table>
<thead>
<tr>
<th>Floor</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASEMENT</td>
<td>0</td>
</tr>
<tr>
<td>FIRST FLOOR</td>
<td>1</td>
</tr>
<tr>
<td>SECOND FLOOR</td>
<td>2</td>
</tr>
</tbody>
</table>

- Diagram of elevator control system
Programming the cyclic program

The current set position is set to the selected floor in the individual steps.

e.g. set_pos = basement_pos

Monitoring and control of the positions is handled outside of the step. If the current position is smaller than the set position, it must be increased until it is equal to the set position. The direction of the movement and the motor must be considered.

```
if ( act_pos > set_pos ) then
    motor = 1 ; Switch motor on
    direction = 0 ; Downwards directions
    act_pos = act_pos - 1
else if ( act_pos < set_pos ) then

; the program should be completed here
```

If the set position is the same as the actual position, then the motor should be switched off.

Project Name: proj_ab
Task Name: elevator
Resource: ?
2.9 Loops

Loops working together with a condition, allow one or more statements to be executed repeatedly. In Automation Basic, all types of loops are created with a single construct.

### Loop Variant 1

<table>
<thead>
<tr>
<th>Key Words</th>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>LOOP i := 0 TO 4 DO</td>
<td>Introduction</td>
</tr>
<tr>
<td>TO/DOWNTO</td>
<td>res := value + i;</td>
<td>Statement block</td>
</tr>
<tr>
<td>DO</td>
<td></td>
<td>Loop End</td>
</tr>
<tr>
<td>ENDLOOP</td>
<td>ENDLOOP</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.8: Loop variant 1

The statement block is only executed when conditions are not met.

### Loop Variant 2

<table>
<thead>
<tr>
<th>Key Words</th>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>LOOP</td>
<td>Introduction</td>
</tr>
<tr>
<td>EXITIF</td>
<td>EXITIF res &gt; 2000; res := value + i;</td>
<td>Exit Condition</td>
</tr>
<tr>
<td>ENDLOOP</td>
<td>ENDLOOP</td>
<td>Statement block</td>
</tr>
</tbody>
</table>

Table 8.9: Loop variant 2

The statement block is executed at least once.

### Loop Variant 3

<table>
<thead>
<tr>
<th>Key Words</th>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP</td>
<td>LOOP</td>
<td>Introduction</td>
</tr>
<tr>
<td>EXITIF</td>
<td>res := value + i;</td>
<td>Statement block</td>
</tr>
<tr>
<td>ENDLOOP</td>
<td>EXITIF res &gt; 2000; ENDLOOP</td>
<td>Exit Condition</td>
</tr>
</tbody>
</table>

Table 8.10: Loop variant 3

**IMPORTANT**

Make sure that endless loops are not created because they will cause a cycle time violation.
Example

In a house, the temperature is measured at 10 different locations. To improve monitoring of the heating costs, the average value should always be calculated.

The solution should be created using a loop !!!

If a temperature is outside the limits,

\[
\begin{align*}
\text{TEMP\_MAX} & = \quad 80 \, ^\circ \text{C} \\
\text{TEMP\_MIN} & = \quad 10 \, ^\circ \text{C}
\end{align*}
\]

then an error bit should be set.

Project Name: proj_ab
Task Name: temp_1
Resource: C#1
2.10 Select Statements

Select statements allow sequential control to be easily programmed and later monitored.

We will clarify sequential control programming using the following electrically operated punch machine.

<table>
<thead>
<tr>
<th>Select Statement</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>SELECT sStep</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Variable := 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WHEN StopKey = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cmdMotor := 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEXT DELAY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATE</th>
<th>state DELAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHEN</td>
<td>cmdMotor := 0</td>
</tr>
<tr>
<td>NEXT</td>
<td>UpKey = 1</td>
</tr>
<tr>
<td></td>
<td>NEXT UPWARDS</td>
</tr>
<tr>
<td></td>
<td>DownKey = 1</td>
</tr>
<tr>
<td></td>
<td>NEXT DOWNWARDS</td>
</tr>
</tbody>
</table>

| state UPWARDS    | WHEN EndTop = 1 |
|                  | NEXT DELAY |
|                  | cmdMotor := 1 |
|                  | CmdDirection := 1 |

| state UPWARDS    | WHEN EndBottom = 1 |
|                  | NEXT DELAY |
|                  | CmdMotor := 1 |
|                  | CmdDirection := 0 |

<table>
<thead>
<tr>
<th>ENDSELECT</th>
<th>ENDSELECT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>End of loop</td>
</tr>
</tbody>
</table>

Tab. 8.11: Select statement
2.10.1 Syntax Descriptions

- A select statement can consist of any number of states!

- **One state** is executed per task cycle!

- The first state in the select construct has number 0.

- For each **next** statement, there must be a **corresponding state**!

- If an exit condition is met, the following commands are no longer executed!

- Number code of the state (Data type: UINT, Scope: local)
  The **step number** which is always visible and the best possible **monitoring** which results from it. Writing to this PV can also influence the process.

- **Global statements** and transfer conditions
  They are executed in each cycle and can therefore be used for error monitoring and to quickly react to special events and high priority queries.

- The **structure** of the Select construct **must** be used, i.e. a state cannot be exited using an IF statement!

Example

```
WHEN LowEnd = 1
  motor = 0
  NEXT DELAY
```

- **Nesting** Select statements is possible!

- **Outputs** should only be written to **once**!
Example

Create a program in AB for the following chemical system. The solution should be created using a Select statement.

Description of the process:

**Press start...**
Open water valve

**Water OK reached...**
Close water valve, Switch on stirring mech., Open color valve

**Sensor Full reached...**
Close color valve, Wait 30 sec. Open drain valve Switch on drain pump

**Sensor Low reached...**
Switch off pump Switch off stirring mech. Close drain valve

- Program the individual steps in AB
- Test the program
- How should E_STOP handling look?

Project Name: proj_ab
Task Name: mix_lq
Resource: ?
2.11 Working with Function Blocks

2.11.1 FBK Call

The function blocks are called like commands. They are accessed using their name. In brackets, then input and output variables follow in order beginning with the first input.

Example of a 2 second turn-on delay:

(* Function block call *)

\[ \text{Preset} := \text{T#2s;} \]
\[ \text{TON(Input, Preset, Off, Elapse)} \]

or:

\[ \text{TON(Input, T#2s, Off, Elapse)} \]

IMPORTANT
All input and output parameters must be entered!

2.11.2 Alias FBK Call

The main difference between an alias call and the procedure used previously is that values are assigned using a freely selectable structure name / alias name and structure elements with the same name as the FBK parameters.

Example: Alias call for TON function block (Library: Standard)

\[ \text{time} = \text{T#2m_30s_500ms} \]
\[ \text{TON_xx.IN := Input} \]
\[ \text{TON_xx.PT := time} \]
\[ \text{TON_xx FUB TON()} \]

\[ \text{Elapse := TON_xx.ET} \]
\[ \text{Output := TON_xx.Q} \]
Example

Use a 16-bit up/down counter

The count procedure should be carried out by the CTUD function block (Library: Standard).

The CTUD FBK requires the following parameters:

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CU</td>
<td>BOOL</td>
<td>Up counter input</td>
</tr>
<tr>
<td></td>
<td>CD</td>
<td>BOOL</td>
<td>Down counter input</td>
</tr>
<tr>
<td></td>
<td>RESET</td>
<td>BOOL</td>
<td>Reset counter to 0</td>
</tr>
<tr>
<td></td>
<td>LOAD</td>
<td>BOOL</td>
<td>Load counter with preset value</td>
</tr>
<tr>
<td></td>
<td>PV</td>
<td>UINT</td>
<td>Preset value and compare value</td>
</tr>
<tr>
<td></td>
<td>QU</td>
<td>BOOL</td>
<td>TRUE, if CV &gt;= PV</td>
</tr>
<tr>
<td></td>
<td>QD</td>
<td>BOOL</td>
<td>TRUE, if CV &lt;= 0</td>
</tr>
<tr>
<td></td>
<td>CV</td>
<td>UINT</td>
<td>Current count</td>
</tr>
</tbody>
</table>

Tab. 8.12: Parameter list for CTUD FBK

The function block is to be called using a normal and an alias call.

Project Name: proj_ab
Program Name: ab_fbk

Resource: ?
# DATA HANDLING

1. OVERVIEW ................................................................................................. 2

2. PROCESS VARIABLES .............................................................................. 3
   
   2.1 General Information ........................................................................... 3
   
   2.2 Data ....................................................................................................... 4
   
   2.3 Basic Data Types ................................................................................... 5
   
   2.4 User Data Types .................................................................................... 7
   
   2.5 Function Block Data Types ................................................................. 17
   
   2.6 Dynamic Process Variables ............................................................... 20

3. DATA MODULES ..................................................................................... 28
   
   3.1 General Information ........................................................................... 28
   
   3.2 What is a data module ? ......................................................................... 29
   
   3.3 What advantages does a data module offer ? ....................................... 30
   
   3.4 Creating a Data Module in AutomationStudio ..................................... 31
   
   3.5 Reading a Data Module from the Application ..................................... 34
   
   3.6 Creating and Writing to a Data Module from the Application 38
   
   3.7 Autonomous Data Module Memory ................................................. 40

4. MEMORY MANAGEMENT ......................................................................... 41
   
   4.1 General Information ........................................................................... 41
   
   4.2 Memory Access .................................................................................... 42
   
   4.3 Location .................................................................................................. 43
   
   4.4 Memory Organization ........................................................................... 47
1. OVERVIEW

Process Variables
For optimal control and separation of information, the user is provided various “Basic Containers” to store data. They can be combined as desired. Simple and flexible access possibilities are available.

Data Modules
Information should often be clearly structured as tables, lists, recipes. Compact and simple access possibilities are provided.
Data modules have overwrite protection.

Memory
Different conditions make it necessary for programmers to store data using suitable memory media with respective physical properties.
2. PROCESS VARIABLES

2.1 General Information

Basic Data Types
BOOL, USINT, UINT, UDINT, SINT, INT, DINT and REAL with an array length of one in the variable declaration.

User Data Types
Organization of data in clear units, arrays, structures, arrays of structures.

Function Block Data Types
Grouping input/output data and internal FBK information using alias PVs.

Dynamic Process Variables
A powerful tool used for simple solutions to complex applications.
2.2 Data

We will begin with the question: “What do we want to save?”

- Program code
- Recipes
- Tables
- Machine options
- **Process variables**
- IO information
- Internal variables
- Parameters
- Status values
- ...
- ...

This is information with very different uses for the contents.

In order to describe this information using a common term, we will refer to it as data to be processed.

This data is stored in memory.

**Memory** is the sum of all **containers** where information can be placed.
2.3 Basic Data Types

2.3.1 What is a basic data type?

Our data can be placed in memory. This consists of a certain number of consecutive units.

Such a unit consists of 8 bits and is referred to as a byte or USINT according to IEC61131.

Each of these USINTs = memory locations, has a unique number similar to the address of a house. Using this address, the memory address, AutomationRuntime or our application can access the content of this memory address (read or write).

In our programs, we do not use direct addresses. Instead we use a symbolic names as synonyms for the addresses (PV names).

A memory location interpreted as USINT has a value range from 0...255, or interpreted as SINT from -128..0..+127.

If the information we want to save in memory has a larger values, we have to separate it over several memory locations and interpret it accordingly.

To do this, the compiler provides different container sizes, the basic data types defined in the IEC61131 standard.
2.3.2 How do I use basic data types?

In the declaration, the AutomationStudio is informed of how much memory is should reserve in AutomationRuntime for a static PV.

When using a PV which is not yet recognized in AS, the programmer is requested to select a suitable data type in the declaration using Project: Create or Project: Create all.

![Fig. 9.3: Selecting a basic data type](image)

**Note**

Relationship between type and bit size:

- **SINT** .. 8 BIT  “S” stands for short
- **INT** .. 16 BIT  No special format
- **DINT** .. 32 BIT  “D” stands for double
2.4 User Data Types

Arrays

An array is the grouping of PVs with the same data type that belong together. This grouping has two major advantages:

- One variable name, access various contents using index
- The PVs are all aligned consecutively and can therefore be easily accessed using a pointer

A string is a special form of array.
A string is a zero terminated USINT array.

Structures

A structure is the grouping of PVs with different data types that belong together. This grouping has some major advantages:

- Help with the organization of complex data sets
- The PVs are all aligned consecutively and can therefore be easily accessed using a pointer

Arrays of Structures

A combination of arrays and structures. If a structure is needed often e.g.: positioning several axes, an array is placed over it and the structure is applied several times consecutively.
2.4.1 What is an array?

If you need to manage several units of information with the same size (same data type) that belong together, an array is the simplest solution.

If e.g.: several temperatures are to be saved as INT, an array with a PV name and a corresponding number of elements can be used as user data type instead of individual variables.

The selection of the desired elements takes place using an index between square brackets. This index provides programming advantages when handling this array variable e.g.: access using loop commands!

---

**Fig. 9.4: User Data Type Array**
2.4.2 How do I create an array?

In the declaration, the programmer defines the data type of the PV and also the array length which corresponds to the number of elements desired.
2.4.3 Tips for Arrays

- The largest index possible is always the array length - 1!
- Begin with array index 0.

Note

The defined array range can be exceeded by entering an index that is too large.

In the following example, the reserved memory for the array PV “pvusint_array“ exceeded and a PV which may follow is overwritten!

```
pvusint_array[5] := 123;
```

Such errors (which are difficult to find) should be avoided by careful programming!!

<table>
<thead>
<tr>
<th>From Library</th>
<th>Function Name</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>memcpy()</td>
<td>Copying an array, length is to be entered explicitly</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>sizeof()</td>
<td>Determine array length</td>
<td></td>
</tr>
</tbody>
</table>

Tab. 9.1: Corresponding functions
2.4.4 Strings are also Arrays

- A string is a special type of array.
- A USINT is required for each character.
- Strings are to be declared in the variable declaration as USINT arrays.
- The end of a string is coded using a 0 in the last element (zero termination).
- Therefore an additional array element must be reserved for zero termination.

<table>
<thead>
<tr>
<th>From Library</th>
<th>Function Name</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>strcpy()</td>
<td>Copy a string up to zero termination</td>
</tr>
<tr>
<td>AsString</td>
<td>strcat()</td>
<td>Combine two strings</td>
</tr>
<tr>
<td>AsString</td>
<td>strlen()</td>
<td>Determine string length</td>
</tr>
<tr>
<td>AsString</td>
<td>itoa()</td>
<td>Convert an integer value to a string</td>
</tr>
<tr>
<td>SYS_LIB</td>
<td>DIS_str()</td>
<td>Output a string on the 2010 status display</td>
</tr>
<tr>
<td>SYS_LIB</td>
<td>DIS_chr()</td>
<td>Output a character on the 2010 status display</td>
</tr>
<tr>
<td>SYS_LIB</td>
<td>DIS_clr()</td>
<td>Clear the status display on the 2010</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str3= concat(str1,str2)</td>
<td>Combine 2 strings after str3</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str2=delete(str1,L,P)</td>
<td>Delete L character from str1 beginning at P</td>
</tr>
<tr>
<td>STANDARD</td>
<td>P=find(st1,str2)</td>
<td>Find position P of string str2 in string str1</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str3=insert(str1,str2,P)</td>
<td>Insert str2 after str1 at position P, save in str3</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str2=left(str1,L)</td>
<td>L on left of str1 character to str2</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str2=right(str1,R)</td>
<td>R on right of str1 character to str2</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str2=mid(str1,L,P)</td>
<td>L character from position P from str1 to str2</td>
</tr>
<tr>
<td>STANDARD</td>
<td>L=len(str1)</td>
<td>Determine string length L of str1</td>
</tr>
<tr>
<td>STANDARD</td>
<td>str3=replace(str1,str2,L,P)</td>
<td>Replace L characters starting at position P from str1 to str2</td>
</tr>
</tbody>
</table>

Tab. 9.2: Corresponding functions
2.4.5 What is a structure?

Units of information with the same size or even with different sizes often belong together e.g.: the date elements Day, Month and Year belong together.

The structure user data type can be used to clearly show that these elements belong together in the program code.

A structure consists of basic data type elements which belong to a higher level PV, the PV with data type structure. These elements are separated from the higher level PV by a ".".

![Fig. 9.6: User Data Type Structure](image)
2.4.6 How do I create a structure?

Structure data types are clearly handled in a separate editor. Using menu item **Open: Data Types**, a new structure can be created using the toolbar and its elements can be entered or changed.

![Fig. 9.7: Handling structures](image)

In the declaration, the programmer selects the structure type:

![Fig. 9.8: Selecting the structure data type](image)
2.4.7 Tips for Structures

The compiler automatically (implicitly) follows the memory access rules when reserving memory for PVs required by AutomationRuntime or AutomationTarget.

An example of such a rule is:
PVs with a data type larger than USINT always begin in a structure on an even memory location!

Therefore this rule, which is called alignment behavior of the compiler in the programming language, must be followed by the compiler during memory management.

If necessary, the compiler will insert filler bytes in the structure. However, they remain hidden from the user i.e. memory is reserved but cannot be directly accessed using a PV name.

Note

To make these filler bytes visible, the programmer should handle the alignment behavior explicitly using reserve bytes.

Advantages:

- If the structure needs to be expanded later, a reserve byte can be used without changing the structure size!
- On different platforms, different alignment rules could be required. Explicit alignment allows platform independent, portable code!

<table>
<thead>
<tr>
<th>From Library</th>
<th>Function Name</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsString</td>
<td>memcpy()</td>
<td>Copy a structure, length is to be entered explicitly</td>
</tr>
<tr>
<td>AsString</td>
<td>sizeof()</td>
<td>Determine structure length</td>
</tr>
</tbody>
</table>

Tab. 9.3: Corresponding functions
Example

An axis should be positioned.

The following data is required for positioning:

- Name [USINT]
- Target position [INT]
- Speed [INT]
- Acceleration [INT]
- Active [USINT]

Organize the elements under a common topic as a structure and then use it as data type for a PV that you assign values to in the INIT SP of the program.

Check the results in Watch!

Project Name: Data
Program Name: dt_ex
Resource: C#4
2.4.8 Combination of Array and Structure

In the declaration, the programmer selects a structure data type and defines the array length for the desired number of elements.

Fig. 9.9: Selecting the structure data type with array length
2.5 Function Block Data Types

Function blocks process input data and create output data using internal variables. In order to have a clear overview of the data, it can be organized with a structure. The data type is provided to the programmer by the LibraryManager in AutomationStudio according to the respective FBK.

2.5.1 Declaration of an Alias PV

In the declaration, the programmer has to select the data type belonging to the FBK. Using the data type, the compiler recognized which FBK code should be called for text-based IEC languages and AB.

A PV with function block data type is called an Alias PV.

Fig. 9.10: Selecting a function block for an Alias PV
The FBK call can be made from the main menu item **Insert: Function**.

![Inserting a FBK call](image-url)
Example

Controlling the external fan for a spindle motor.

When switching the spindle motor off by resetting the PV `gDoMotor` to 0, the fan motor, which is controlled by the PV `gDoFan`, should continue running for 10 seconds.

Program a turn-off delay using the TOF FBK in AB and check the behavior.

Project Name: Data
Program Name: dt_ex
Resource: C#4
2.6 Dynamic Process Variables

2.6.1 What is a dynamic process variable?

The PVs used up till now have a fixed location in memory. The content of these memory locations can be referenced by name and processed.

If you want to access a different memory location, you have to use the name of the PV that is assigned to this memory location.

When compiling, this PV is assigned a fixed address which cannot be influenced by the user during runtime.

These types of PVs are called static variables!

However, a PV name should often be used that can access different memory locations. To do this, the PV is declared as pointer and a fixed address is not assigned when compiling!

The memory location that the pointer should access can be defined by the user during runtime, this is referred to as “defining a reference for the pointer”.

These types of PVs are called dynamic variables!

The content of the memory locations is interpreted according to the selected data type beginning at the start address. The data type of the dynamic PV functions as a mask.

Fig. 9.12: Dynamic process variables
2.6.2 Determining the Address

AutomationStudio assigns each static PV an address offset which is linked to free memory addresses on the AutomationTarget by AutomationRuntime.

The memory address for a PV can be determined in the application program in language AB using the operator “adr()”.

It returns the address as a UDINT value.

```plaintext
;** Example **
Adr_PvInt:= adr(PvUint)
```

![Fig. 9.13: Determining the address of a PV using operator adr()](image)
**Example**

Create an array of axis structures with array length 3 to provide X,Y,Z axis coordinates for a CNC system.

Analyze the division of memory for the axis structure using the “adr()” operator and enter the information in the following sketch.

In the column “Address”, enter the memory addresses that have been determined. In the column “Memory Content”, indicate the alignment fillers using hatch marks in the memory locations. In the column “PV Name”, enter the corresponding structure element.

**Note**

Determining the memory address of a structure element using “adr(Topic.Element)”

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Content</th>
<th>PV Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>16#</td>
<td></td>
<td>axis[0].name</td>
</tr>
<tr>
<td>16#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16#</td>
<td></td>
<td>axis[1].name</td>
</tr>
<tr>
<td>16#</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Fig. 9.14: Memory content*
2.6.3 How do I create a dynamic process variable?

In the declaration, the programmer defines two points:

- **Data type**
  Defines the “mask” used later to interpret the contents of the referenced address

- **Dynamic scope**
  The address offset is assigned to the PV during runtime

![Fig. 9.15: Defining a dynamic PV](image-url)
2.6.4 Access

A dynamic PV can be assigned a memory address during runtime, this procedure is called referencing or initializing.

As soon as a dynamic PV is initialized, it can access the contents of the memory location it is “pointing” to according to the data type.

In the program code, a dynamic PV is used like a static PV after initialization.

![Fig. 9.16: Access](image-url)
Example

Create an array variable with length 30 and data type USINT. Enter the values in the elements using the PV Monitor.

Create a dynamic PV with data type:

- USINT
- UINT
- INT

and “place” it over the array variable as mask.

Begin with the start address of the array variable and then „shift“ the mask by increasing an offset!

Compare the value interpreted using the mask with the value in the array variable!
Example

AutomationRuntime receives positioning data AutomationNet. This data should be saved in an array variable “receive”, data type USINT[30]. Enter the values in the array variable using the PV Monitor.

Create a dynamic PV with the axis data type for an axis and use it as a mask to evaluate the data received from the array “receive”.

Initialize the dynamic PV with the start address of the array PV and handle the selectable data set number using an additive offset.

![Memory mask](image-url)
2.6.7 Programming Techniques for Dynamic PVs

Preparing dynamic PVs provides the programmer all the possibilities needed to create flexible, compact and high performance code for automation solutions.

Using pointers, it is possible to access and evaluate the entire application memory.

It is important to guarantee correct, controlled initialization of dynamic PVs.

Humans make mistakes, and programmers are human so certain steps should be taken when using dynamic PVs to prevent uncontrolled memory manipulation.

Note

Discuss the possibilities for control of dynamic memory access in the group and keep in mind the need to correctly use this powerful tool – “Dynamic PVs”!
3. DATA MODULES

3.1 General Information

What is a data module?

What advantages does a data module offer?

Creating a Data Module in AS

Reading a Data Module from the Application

Creating and Writing to a Data Module from the Application

Autonomous Data Module Memory
3.2 What is a data module?

Data modules make it easier to manage information and increase flexibility. A data module is a memory area which is reserved in a continuous section. The content of this data module (memory area) can be entered using a simple ASCII editor, the Data Module Editor using a user defined mask. The content of the data module can also be written to by the application. The data in the module can be transferred from AutomationStudio to AutomationRuntime or from AutomationRuntime to AutomationStudio!

Fig. 9.18: What is a data module?
3.3 What advantages does a data module offer?

We are already familiar with managing information in process variables which are physically stored in RAM.

The following questions are often asked concerning the special requirements for data management in modern automation applications:

- How can I protect against data loss during a warm start or power failure if a backup battery is not available?
- How can I protect against data loss during a cold restart?
- How can I increase the amount of data in a simple and flexible manner?
- How can I define or limit read and write access rights for a section of my data?

The following flowchart shows possible solutions:
3.4 Creating a Data Module in AutomationStudio

Create a data module in a software tree using **Insert:New Object** and making the selection Data object = Data module.

Like every other module, give the data module a unique name which can be used by the application to read or write data during runtime.

![Insert Object](image)

**Fig. 9.20:** Creating a data module
By double-clicking on the object, an easy to use ASCII editor with syntax coloring is opened to enter data in the data module.

The user enters data or comments here.

Comments are all started with a semicolon and allow flexible construction of a mask, e.g.: represented as follows for a table

Data is separated in a line by a comma or line break.

Data can be entered as follows:

- Numerical in decimal, hexadecimal or binary format
- as ASCII text **without** zero termination between ‘text’
- as ASCII string **with** zero termination between “string”

Numerical values can be entered as whole numbers or real values with decimal point.

**Note**

The memory reserved for the value, the data type, is determined by the value range and the number of characters!
Example

Data module handling

- Create the following data module with the name “table”:

```
;--------------------------
;--- Positioning data---
;+-------------------Positioning variant
 ;| A/R..Absolute/Relative
 ;| +----------------Target pos./Positioning path
 ;| | [DINT]
 ;| | +---------Positioning speed
 ;| | | [UINT]
 ;| | | +---Acceleration
 ;| | | [UINT]
 ;| "A", 00020000, 30000, 10000 ; 1st positioning step
 ;"R", 00000100, 00200, 10000 ; 2nd positioning step
```

- Transfer the data module to the PCC and check if the module is on the PCC.

- Create another data module with the name “t_text”.
  Make sure that the second line is NOT zero terminated.

```
; 1 2 3
;1234567890123456789012345678901234
;
"This is our second data module! "
'This is the second line
```

- Transfer the data module to the PCC and check if the module is on the PCC.

Data module name: table, t_text

Resource: DAT
3.5 Reading a Data Module from the Application

The B&R “Syslib” library provides the function blocks needed to allow simple data module access so the application can read the contents. They can be used as shown in the following diagram:

Fig. 9.22: Reading a data module
Detailed information about function block parameters can be found in the AutomationStudio online help.

Fig. 9.23: Step 1

Fig. 9.24: Step 2

Fig. 9.25: Help information
Example

Write a task to read positioning data from the data module “table”.

- To do this, create a dynamic structure with length 2 over the table.
- The data should be copied to a static PV with positioning structure type.

The following possibilities are available:

```c
memcpy( ...)  
DA_read(...) 
```

Read the respective online information concerning the function blocks and select one of the possibilities.

- Use the INIT SP to determine the data module system information

- Overwrite one of the values from the static structure PV and then from the dynamic structure PV in the PV Monitor and check the reaction of AutomationRuntime!

Program name: da_struc

Resource: C#4
Example

Write a task that reads the following information from the data module “t_text”.

- Copy the first line of your data module to the byte array “ASCII_1” using the function: ”strcpy(target address, source address)”.

- Read the 3rd byte from the data module and write it to the variable “spec_byte”.

- Copy the second line of your data module to byte array “ASCII_2”. To do this, use a loop and the dynamic PV “DYN_PV”.

Program name: da_read
Resource: TC#4
3.6 Creating and Writing to a Data Module from the Application

The B&R “Syslib” library provides the function blocks needed to allow the application to create a data module used to save user parameters during runtime. Write access to the data module takes place in a controlled manner using a write function which automatically carries out the checksum correction. These functions can be used as shown in the following diagram:

![Diagram of Creating, writing to and saving a data module]

Fig. 9.26: Creating, writing to and saving a data module
Example

To practice creating and writing to data modules, write a task that carries out the following actions:

- A data module which is initialized by a structure should be created when a positive edge occurs on an input.

Struktur: init_da

| .USINT_var:    | USINT | 1 |
| .UINT_var:     | UINT  | 1 |
| .USINT_array:  | USINT | 10 (String) |

If the data module was already created, all necessary parameters, e.g.: module length, should be requested from AutomationRuntime.

- The structure has to fit in the data module 3 times.

- The entire data module should be created using initialization values, with values <> 0

- The values are read using dynamic variables to check the content of the data module.

- Check the length of the data module. Does it correspond to the length you expected?

- Change the element UINT_var in the data module when a positive edge occurs on an input. An analog value should be written to the variable.

Program name: da_first

Resource: ??
3.7 Autonomous Data Module Memory

A data module created by the application using DA_create(...) is normally placed in USR RAM so that the data can be manipulated.

If it is also necessary to protect the information in the data module during a warm and cold start, the data module can be copied to USR ROM.

If the information in the data module should be changed again, it must be saved in USR ROM again and the old memory is marked as being bad. Memory that is marked as being bad can be released again by deleting the USR ROM, but all tasks are normally lost.

Therefore, AutomationRuntime provides the application a special area in FLASH (DM_USER_FLASH) for autonomous management of data modules on the following AutomationTargets:

- B&R2003: CPx70, CPx74
- B&R2005: XP152, IF152, IP161, CP260, IF260
- B&R2010: IF100, IF101

This 64 kByte memory area can be manipulated by the user. The user can delete this area of the FLASH independent of the USR ROM.

AutomationRuntime provides the following services in the library “DM_LIB.BR” to manage this memory:

- DMclear() .. Delete DM USER FLASH
- DMstore() .. Store data module in DM USER FLASH
- DMfree() .. Information, how much DM USER FLASH is still free
4. MEMORY MANAGEMENT

4.1 General Information

We will summarize memory management using the following questions.

Memory Access
Who can save data?

Location
Where can data be saved?

Memory Organization
How can information be saved?
Who can reserve and allocate memory?
4.2 Memory Access

Who can read, write and store data?

In general, there are two possibilities for memory access: access by the user and by the system.

User

- Application
  Programs work with PVs
- AutomationStudio
  Install modules, watch, etc.
- Operator + Visualization
  Human-Machine Interface (HMI)

System

- Internal variables/data from AutomationRuntime
- Input data from hardware modules
- Communication using Interfaces

These two types of tasks are very different and therefore have different requirements for storing and accessing data.

Fig. 9.27: Memory access
4.3 Location

We have found out that data are needed by different users to complete their tasks. The data is stored in memory according to the requirements mentioned previously. Two subdivisions exist:

- Physical location
- Logical location

The characteristics and purposes of these two locations will be repeated and summarized in the following sections.

Fig. 9.28: Memory locations
4.3.1 Physical Memory Locations

An AutomationTarget has various physical memory media.

<table>
<thead>
<tr>
<th></th>
<th>modular</th>
<th>onboard</th>
<th>Effect after: Warm Start</th>
<th>Effect after: Cold Start</th>
<th>Special Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>✔️</td>
<td>✔️</td>
<td>All memory locations have the value 0</td>
<td>All memory locations have the value 0</td>
<td>Very fast access</td>
</tr>
<tr>
<td>SRAM</td>
<td>✔️</td>
<td>✔️</td>
<td>Memory is remnant</td>
<td>All memory locations have the value 0</td>
<td>Each bit can be reset individually</td>
</tr>
<tr>
<td>FLASH</td>
<td>✔️</td>
<td>✔️</td>
<td>Memory is remnant</td>
<td>Memory is remnant</td>
<td>Nonvolatile</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Tab. 9.4: Physical memory

Notes

Concerning the terms above:

- modular................ modular user memory, e.g.: Memcard or MExxx
- onboard...............memory is directly in the AutomationTarget module
- remanent.............memory is not changed by AutomationRuntime

A detailed description can be found in the hardware manual, the AutomationStudio online help or on the Internet homepage.
4.3.2 Logical Memory Location

The memory is provided to AutomationRuntime using physical blocks. AutomationRuntime separates the physical memory into logical units. It separates the RAM and FLASH into a User and System area. The User and System areas are also separated into smaller, independent areas.

![Physical and logical memory locations](image-url)
In the following table, the locations are summarized again including typical content and warm and cold start behavior.

<table>
<thead>
<tr>
<th>Name</th>
<th>Content</th>
<th>Effect after Warm Start</th>
<th>Effect after Cold Start</th>
<th>Special Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USR RAM</td>
<td>Program + Data modules</td>
<td>Memory is remanent</td>
<td>All memory locations have the value 0</td>
<td>Data in modules has checksum protection</td>
</tr>
<tr>
<td>USR ROM</td>
<td>Program + Data modules</td>
<td>Memory is remanent</td>
<td>Memory is remanent</td>
<td>Data in modules has checksum protection</td>
</tr>
<tr>
<td>FIX RAM</td>
<td>Program + Data modules</td>
<td>Memory is remanent</td>
<td>Memory is remanent</td>
<td>Cold start safe</td>
</tr>
<tr>
<td>DM_USER</td>
<td>Data modules</td>
<td>Memory is remanent</td>
<td>Memory is remanent</td>
<td>Flash can be organized and deleted by the application</td>
</tr>
<tr>
<td>DPR</td>
<td>Local + global PVs</td>
<td>Memory is remanent</td>
<td>All memory locations have the value 0</td>
<td>Access of PVs also possible from other automation targets &amp; PVI</td>
</tr>
<tr>
<td>Permanent Memory</td>
<td>Freely configurable</td>
<td>Memory is remanent</td>
<td>Memory is remanent</td>
<td>Cold start safe</td>
</tr>
<tr>
<td>Temporary Memory</td>
<td>Freely configurable</td>
<td>All memory locations have the value 0</td>
<td>All memory locations have the value 0</td>
<td>Large memory blocks can be reserved during runtime</td>
</tr>
<tr>
<td>Stack</td>
<td>FBKs &amp; Freely configurable</td>
<td>All memory locations have the value 0</td>
<td>All memory locations have the value 0</td>
<td>For user function blocks</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS ROM</td>
<td>System + Data modules</td>
<td>Memory is remanent</td>
<td>Memory is remanent</td>
<td>Modules are also activated in diagnose mode</td>
</tr>
<tr>
<td>Static RAM</td>
<td>Automation Runtime</td>
<td>All memory locations have the value 0</td>
<td>All memory locations have the value 0</td>
<td>Area protected from the user for logbook, system tables, etc.</td>
</tr>
<tr>
<td>Stack</td>
<td>Automation Runtime</td>
<td>All memory locations have the value 0</td>
<td>All memory locations have the value 0</td>
<td>For system function blocks</td>
</tr>
</tbody>
</table>

Tab. 9.5: Logical memory
4.4 Memory Organization

B&R AutomationRuntime provides various easily implemented methods for storing data.

This allows various requirements to be met e.g.: data with defined access rights.

All AutomationRuntime versions offer the following possibilities:

- Process variables
- Dynamic memory allocation
- Data modules

Process variables allow simple and direct access of data memory using symbolic names.

Dynamic memory allocation allows flexible memory organization during runtime.

Data modules capsule data in protected areas which include all B&R module characteristics e.g.: checksum test, download, upload, data editor in AutomationStudio, etc.

Fig. 9.30: Memory management
Before we can access the memory (use it), someone has to reserve it for us. The memory must be organized.

This is important so that each user receives a separate, assigned area.

Memory can be organized in two ways:

- static
- dynamic

Static memory is automatically reserved by the compiler when using PVs and the relationship between PV name and the reserved memory address is entered in a PV table.

Dynamic memory is requested from AutomationRuntime by the user in the application using FBKs and referenced using a dynamic PV. For programming, requesting memory is referred to as “Allocation”. Dynamic memory can be requested during runtime with a selected size, therefore the user has very flexible memory organization which can be changes as desired during runtime.

Fig. 9.31: Memory organization
TIMING PROCESSING UNIT

1 OVERVIEW .......................................................................................... 2

2 TIMING PROCESSING UNIT .............................................................. 3
   2.1 What is a Timing Processing Unit? ............................................ 3
   2.2 Block Diagram of a Processor with TPU ......................... 3
   2.3 Functions ...................................................................... 4

3 TPU MODULES ........................................................................... 5
   3.1 B&R AutomationTarget 2003 .......................................... 5
   3.2 B&R AutomationTarget 2005 .......................................... 6

4 LTX FUNCTIONS .................................................................. 7
   4.1 Configuration of the Hardware ....................................... 7
   4.2 Use of LTX Functions ....................................................... 8
   4.3 Example Gate Measurement ............................................ 9
1 OVERVIEW

Timing Processing Unit
A Timing Processing Unit is used when creating solutions for time critical applications because it allows certain events to be reacted to in the µs range.

TPU Modules
This chapter provides an introduction to the major parts of a TPU and an overview of the hardware available for such applications.

LTX Functions
Selecting and inserting functions used to operate a TPU in Automation Studio are explained in detail here. Examples are also provided.
2 TIMING PROCESSING UNIT

2.1 What is a Timing Processing Unit?

A Timing Processing Unit, (TPU) is an additional hardware unit which supports the CPU. It can be used to execute simple and also time critical functions without loading the CPU.

With each action that occurs, e.g. positive edge on an input, an LTX function is called. These functions are processed by the TPU. This allows reaction times in the μs range.

LTX ... Logic Timing Functions

2.2 Block Diagram of a Processor with TPU

After selecting the desired functions, the TPU Code Linker in Automation Studios creates code which is placed in TPU RAM during a warm start by the CPU on TPU capable modules.

Then this memory area can only be accessed by the TPU.
2.3 Functions

- Input recognition / input edge counter
- Output comparator
- Pulse width modulation
- Synchronized pulse width modulation
- Period measurement
- Period measurement with edge recognition
- Position synchronized pulse generator
- Stepper motor control
- Gate measurement
3 TPU MODULES

3.1 B&R Automation Target 2003

DI135, DO135, AI261, AI294, AI351, AI354, AI774, AO352, NC161

These screw-in modules can also be operated on the left of the CPU. However, the integration of LTX functions is only possible on the CP interface.

3.1.1 DI135

Features:

- 4 high speed digital inputs 24VDC
- Incremental encoder operation 50 kHz
- Event counter operation 100 kHz
- 1 comparator output 24 VDC

Area of use:

- Period measurement
- Gate measurement
- Incremental encoder / encoder

Typical applications:

- Bottling system, etc.
3.1.2 DO135

Description:
- Digital output module with 4 FET outputs
- Switching voltage 12-24VDC
- Continuous current max. 0.1A
- Max. switching frequency 100 kHz

Area of use:
- Pulse width modulation
- Stepper motor control
- Absolute encoder (SSI)

Typical applications:
- Temperature control for extruders
- Stepper motor control, etc.

Note
All other modules and detailed information are contained in the 2003 User’s Manual.

3.2 B&R Automation Target 2005

- IP151
- IP152
- IP161
- IP350

All of these modules have high speed analog / digital inputs and outputs which can be accessed using TPU functions. Detailed information and areas of use can be found in the 2005 User’s Manual.
4 LTX FUNCTIONS

4.1 Configuration of the Hardware

In order to work with LTX functions, the required hardware modules must be inserted in AS. The hardware definition can take place in two ways.

- Load hardware configuration from target system
  This method can always be used,
  if the hardware is already available during project creation.

- Insert hardware manually

![Inserting hardware in AS](image)

**Fig. 10.2:** Inserting hardware in AS
4.2 Use of LTX Functions

AS provide the user with a large selection of LTX functions. After inserting such a function block, code is generated which is stored in the folder CPU -> System. This code is copied to TPU RAM after a warm start.

4.2.1 Inserting LTX Functions

To integrate this FBK, select the TPU tab.

Note

The TPU tab is only available for modules with TPU functionality.
4.3 Example Gate Measurement

The frequency of a signal should be determined.

The signal comes from a digital output which toggles and is connected with the DI135 input.

- Select a suitable LTX function block
- Calculate the frequency in a timer task class
- Create task with toggle output

Project Name: tpu_pro1
Task Name: frq_cnt
Resource: T#1

Task Name: toggle
Resource: ???
4.3.1 Gate Measurement Instructions

Fig. 10.5: Insert an LTX FBK

Fig. 10.6: Select the LTX function
Note

After selecting the FBK, select the CPU tab again for programming. In this way, the code for this function is automatically generated and integrated in CPU->System. The extended help can be opened by pressing the F1 key with the TPU tab selected.

Additionally, a library is imported where the FBKs used are entered. Now it is possible to insert the function blocks in the program.

Inserting a Function in LAD

Calculating the Frequency:
Using the FBK for gate measurement, the frequency of the signal can now be calculated.

Note

The entire solution can be found in help
LIBRARY MANAGER

1 OVERVIEW ........................................................................................................... 2
  1.1 Functions ...................................................................................................... 3

2 LIBRARY MANAGER .................................................................................. 4
  2.1 General Information ........................................................................... 4
  2.2 Term Definitions ................................................................................ 5
  2.3 Library Guidelines ............................................................................. 6
  2.4 Global Settings .................................................................................... 8

3 B&R LIBRARIES .................................................................................. 9
  3.1 Overview of Standard Libraries ....................................................... 9
  3.2 Online Help .......................................................................................... 10
  3.3 Insert Library ....................................................................................... 12

4 USER LIBRARIES .................................................................................. 15
  4.1 Creating a Library ................................................................................ 15
  4.2 Creating an IEC Function Block .................................................... 18
  4.3 Function Block Properties ............................................................... 19
  4.4 FBK Interface ..................................................................................... 20
  4.5 Source Code for the Function Block ........................................... 21
  4.6 Creating Online Help ....................................................................... 24
  4.7 Creating a C Library .......................................................................... 24
  4.8 Reusing Libraries ............................................................................... 25

5 PG2000 LIBRARIES ............................................................................... 27
  5.1 PG2000 Porting Guide ....................................................................... 27
1 OVERVIEW

Library Manager
   The Library Manager is used to manage all libraries used in a project. This chapter provides a detailed description of the characteristics and possibilities of this tool.

B&R Libraries
   This chapter contains an overview of the B&R Standards Libraries and an explanation of how they can be inserted in a project and managed.

User Libraries
   In addition to the standard B&R functions, the user also has the possibility to create function. These functions can be grouped in libraries.

PG2000 Libraries
   How can I add PG2000 Libraries to AS?
1.1 Functions

1.1.1 Advantages of Functions

Automation Studio provides many standard functions for the user. It is also possible to create functions. The use of functions has the following advantages.

- **Saves time**
  It is possible to use existing functions instead of having to create them.

- **Programs are clearer and easier to service**
  The program code contains parameters which are easy to understand and check.

- **Prevents unnecessary errors**
  B&R standard function blocks are already tested, which prevents possible typing errors or mistakes.

- **Standardization of complex tasks**
  Large or commonly used program sections (positioning, control algorithms, etc.) are thought through once and written in a form that allows users to easily define parameters later.

- **Multiple usage in further projects**
  A function is often used multiple times in a project which allows the solution to be standardized and simplified.
2 LIBRARY MANAGER

2.1 General Information

The Library Manager (LibMan) is used to manage all libraries which are integrated and defined in project.

With Library Manager, you can:

- Insert B&R standard libraries in a project
- Insert other libraries in a project
- Create and manage libraries
- Manage libraries created in PG2000 with AS

The Library Manager is started using the menu item **Open: Library Manager**.
2.2 Term Definitions

Function
A function is a program organizational unit which returns exactly one value. A function has one or more inputs but only one output. Therefore it can be called in programs in a high level language directly as operand.

\[ \text{e.g.: } \text{if } \text{edgepos}(\text{gDiMotorStart}) = 1 \text{ then} \]

Function Block
The FBK is a program organizational unit which returns one or more values. It has one or more inputs and outputs.

Component
Function or FBK

Library
A library is a group of several components.

B&R Library
The libraries supplied by B&R which are used for B&R system software and hardware functions are described as standard libraries.

User Library / Third Library
Library created by the user or by a third party.

Binary Library
A library without source text.

Source Library
A source library contains the source text for the components. The source code can be changes at any time.

IEC Library
A library with components which are written in the languages B&R AB, ST, etc. When creating such libraries, one of these of these languages can be used for each component.

C Library
A library which is only written in ANSI C. All components are coded in this language.
2.3 Library Guidelines

Libraries are designed to be used again and again. Therefore it is especially important that they are well planned. This chapter shows a short section of the library guidelines. The complete chapter is contained as an appendix.

2.3.1 Assigning Library Names

In order to make libraries easier to identify and service, they should be assigned clear and meaningful names.

- A library should start with three characters which clearly identify the designer (company) of the library. This character combination can consist of letters and/or numbers.
- This character combination should be used for all libraries from the company.
- Library names are presently limited to a maximum of 8 characters and are defined as follows:

\[ \text{dddLllll} \]

To improve clarity, the part of the name after the designer should begin with a capital letter. The company code should begin with a small letter so that library names also comply with the format for for variables.

**Examples**

```
br_Arith
br_ .. Code for B&R
Arith .. Name of the library
brTrRGL
br .. Code for B&R
Tr .. Code for training
RGL .. Code for control components.
```

RGL is written in capital letters because this will also be the code for the components.
2.3.2 Assigning Component Names

Assigning function block and function names plays a very important role in the appearance of a library. The names should indicate the library that the component belongs to and also has to indicate the functionality.

- Principally, 32 characters are available in AS for component names.
- The first word indicates the library. At least two characters and a maximum of four characters must be used.
- All components in a library begin with the same characters e.g.: ctrlValve, ctrlMotor, etc.
- Only the first letter of the individual words are capitals
- The functionality is then written in a clear form

Examples: ArithSum,

ctrlHeatValve

2.3.3 Version Management

Version numbers should be assigned to all libraries. The following format should be used for the version entry.

The version number consists of four characters.

\[ x.yy.ß \]

- Increased by one for large changes. (yy becomes 0)
- Increased by one for all changes
- Used for Beta versions
2.4 Global Settings

General LibMan settings can be made for the project using the menu item **Project: Settings**. The dialog box can also be reached using the menu item **Edit: Properties** if a library is selected in the left LibMan area.

- **Standard directory**
  Shows the valid library standard directory. The libraries for the project are be taken from here. The path is automatically set by AS and depends on the defined operating system version.

- **Library Directories**
  Additional directories can be entered where LibMan can search for libraries.

- **Standard Target Memory for Libraries**
  The target memory for libraries can be set here.
### 3 B&R LIBRARIES

#### 3.1 Overview of Standard Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASControl</td>
<td>Support of hardware modules</td>
</tr>
<tr>
<td>ASMath</td>
<td>Mathematics functions not covered by the Operator library</td>
</tr>
<tr>
<td>ASString</td>
<td>Functions for memory manipulation and string handling</td>
</tr>
<tr>
<td>BRSystem</td>
<td>Functions for CPU operation</td>
</tr>
<tr>
<td>C220man</td>
<td>Functions for panel controller operation</td>
</tr>
<tr>
<td>CAN_lib</td>
<td>Functions for CAN controller operation</td>
</tr>
<tr>
<td>CANIO</td>
<td>Functions for B&amp;R2003 CAN node operation</td>
</tr>
<tr>
<td>Convert</td>
<td>Conversion functions according to IEC61131-3</td>
</tr>
<tr>
<td>DM_lib</td>
<td>Storage of data modules in nonvolatile memory</td>
</tr>
<tr>
<td>DRV_3964</td>
<td>3964R protocol</td>
</tr>
<tr>
<td>DRV_mbus</td>
<td>Modbus protocol</td>
</tr>
<tr>
<td>DRV_mn</td>
<td>MiniNet protocol</td>
</tr>
<tr>
<td>DVFrame</td>
<td>Frame driver library for serial interface operation</td>
</tr>
<tr>
<td>FDD_lib</td>
<td>Serial floppy drive operation</td>
</tr>
<tr>
<td>IF361</td>
<td>Operation of IF361 interface module (Profibus DP Slave)</td>
</tr>
<tr>
<td>IF661</td>
<td>Operation of IF661 interface module (Profibus DP Slave)</td>
</tr>
<tr>
<td>INAclnt</td>
<td>INA2000 client communication</td>
</tr>
<tr>
<td>IO_lib</td>
<td>Functions for I/O module operation</td>
</tr>
<tr>
<td>NET2000</td>
<td>NET2000 protocol</td>
</tr>
<tr>
<td>Operator</td>
<td>IEC61131-3 standard functions</td>
</tr>
<tr>
<td>PB_lib</td>
<td>Profibus protocol (FMS)</td>
</tr>
<tr>
<td>PPdpr</td>
<td>Functions for exchanging data between CPU and PP</td>
</tr>
<tr>
<td>RIO_lib</td>
<td>Functions for remote I/O operation</td>
</tr>
<tr>
<td>Runtime</td>
<td>Functions for internal support</td>
</tr>
<tr>
<td>Spooler</td>
<td>Allows spooling of data on IPs</td>
</tr>
<tr>
<td>Standard</td>
<td>IEC61131-3 standard functions</td>
</tr>
<tr>
<td>SYS_lib</td>
<td>Various system functions</td>
</tr>
<tr>
<td>TCPIPIMG</td>
<td>Functions for exchanging data using UDP or TCP</td>
</tr>
</tbody>
</table>

Tab. 11.2: Library overview
3.2 Online Help

In AS, an online help system is available for the user with detailed descriptions of all libraries. The help can be opened by pressing the F1 key in LibMan.

The online help offers the user various possibilities to search for topics.

- **Contents**
  Shows an overview of the help topics. Clicking on the main topics opens the lower level topics. Figure 11.3 shows a sample of this page.

- **Index**
  Makes it possible to search for functions or topics.

- **Search**
  Search for certain terms in all topics.

- **Favorites**
  The path to commonly used help topics can be saved here. Simply click on the topic and the corresponding help page is opened.
Example

Open the online help and search for information concerning the function block TON_10ms.
3.3 Insert Library

Using the menu item **Insert: Library** or by clicking on the symbol , the required dialog box is opened.

The list only shows libraries found in the standard directory and library directories. However, it is possible to search for libraries in other directories using the **Browse** button. The dialog box that is opened is used to navigate in the directory tree. The “OK” button is only activated when a valid library structure is found in the directory.
If you select the library on the left side, the following tabs are shown on the right side:

- **Data Types**
  The data types that come with the library are shown here. They can then be used throughout the entire project.

- **Constants**
  The constants required by and included with the library

![Fig. 11.12: Data types/constants](image1)

If a component is selected from a library (see Fig. 11.11), then the respective variable declaration and I/O assignments are shown on the right side.

If detailed information is needed concerning a function block, it can be found in the online help. If the FBK is selected, then the respective page is shown immediately.

![Fig. 11.7: Online help for TON](image2)
Additionally, B&R provides customers the possibility to view an example program for inserting the respective component. This is done as follows.

- Search for the component in the online help using the Index tab (see Fig. 11.7)
- Now the program example can be selected in the desired programming language by pressing the Enter key

![Fig. 11.8: Select programming language](image)

The example shown can then be inserted into the project using copy and paste. The help can be printed at any time.
4 USER LIBRARIES

4.1 Creating a Library

LibMan makes it possible for the user to create libraries, in addition to the existing B&R libraries.

4.1 Inserting a Library

After opening LibMan, the dialog box can be opened using menu item **Insert: Library**, or by clicking on the respective symbol. After activating “New Library”, the name of the new library can be entered in the input field. It must be a name that was not yet used for any other purpose in the entire project. If a name has already been used, it will not be accepted.

The programming language to be used to create the components in the new library also has to be selected. You have to select if you want to use **ANSI C** or one of the **IEC languages**, including B&R Automation Basic to program the components.

![Insert Library dialog box](image)

*Fig. 11.9: Insert Library dialog box*
4.1.2 Library Properties

Properties for a library can be reached using menu item **Edit: Properties**. The desired library must be selected on the left side. For source libraries, the dialog box is used to enter and change these properties. For binary libraries, it is only used to display the properties.

If the source code is available in the project, the parameters can be changed. Otherwise the current settings are displayed.

Only the “General” properties are to be set for IEC libraries. For libraries written in ANSI-C, this dialog box contains further registers which will be described later.

**Description:**
A short description of the library should be entered here.

**Header File:**
Name of the *.h file created for a library. Preset to the name of the library for a new library.

**Version:**
Version number

**Target Platform:**
Target platform for which the library was or should be created.
4.1.3 Library Parameters

The parameters can be seen when the desired library is selected on the left side of LibMan.

- **Data Type**
  Here, user data types belonging to the library can be added by clicking on the symbol or using the <Insert> key. However, they must be created first using the menu item **Open: Data Types** so that they can be selected. If a variable is declared as a structure in a library component, this user data type is automatically added.

- **Constants**
  The constants needed for the library are shown here. Additional constants can be added by clicking on the symbol or using the <Insert> key.

- **Additional Dependencies**
  When adding a library that other libraries have declared as dependent this will automatically be inserted in the project. When using B&R standard functions, the libraries are automatically added as a dependency. However, only the system module e.g.: standard.br for the library will be copied to the software tree of the project desktop here.
4.2 Creating an IEC Function Block

Before a component can be created, a library must be inserted.

Then select the library that has been created on the left side of LibMan where the component should be added. A new component can be added to the library using the right mouse button or the symbol.

Fig. 11.13: Inserting a function / function block

Fig. 11.14: Inserting Sum FBK
The new component is now shown in LibMan.

4.2.1 Selecting the Language

With IEC libraries, the user can select between languages STL, ST and AB.

4.3 Function Block Properties

A short “public” description can be added for each component using the menu item **Edit: Properties**
4.4 FBK Interface

The interface for a component is principally the same as the variable declaration in cyclic program sections. The required variables can be entered here. Unlike a “normal variable declaration”, there are additional possibilities to declare a variable here.

- **VAR_INPUT**
  Input parameters

- **VAR_OUTPUT**
  Output parameters

- **VAR**
  Static variables / FBK local

- **VAR_DYNAMIC**
  Dynamic variable in Automation Basic. This variable is only valid in the FBL and has no affect on outside activities.

- **VAR_INPUT_DYNAMIC**
  Dynamic input/output parameters:
  They are assigned the respective pointer by the ADR function. That means that an address must be connected to this input. Access using the pointer takes place automatically in the FBK.

**Note**

If variables are used in the component source code which are not yet entered in the declaration, the “Auto Declaration Dialog Box” is shown.
4.5 Source Code for the Function Block

To enter the source code, simply select the desired component on the left side of LibMan and press the <Enter> key.

![Fig. 11.16: Selected FBK](image)

The editor will be opened according to the language selected for the component. The code for the FBK can now be entered.

![Fig. 11.17: Source code for the Sum FBK](image)

After closing the editor, the component is saved and is complete.

Now the component can be used in the project just like the standard components.
4.5.1 Heating Example I

The temperature of a room should be monitored.

If the actual temperature “temp_act” is higher than the set temperature “temp_set”, then the variable “cooling” should be set to one.

If the actual temperature “temp_act” is lower than the set temperature “temp_set”, then the variable “heating” should be set to one.

Heating control should be integrated in a FBK which is then called in a LAD task.

- Creating the heating control FBK in B&R AB

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp_act</td>
<td>INT</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>temp_set</td>
<td>INT</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>heating</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>cooling</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
</tbody>
</table>

Fig. 11.18: FBK Interface

- Using the FBK in a LAD task

FBK Name: PRGHeating
Task Name: test_fbk
Resource: ?
Project: libman.pgp
4.5.2 Fill Level Monitoring

**Procedure:**
A container should be filled with a liquid. The fill level constantly changes when adding the liquid, therefore make sure that the valve is only closed when the fill level is over the switch off level for at least 10 seconds.

**Diagram:**

- Handle monitoring in a FBK
- Test the FBK in B&R AB

FBK Name: PRGSwimmer
Task Name: ?
Project: libman.pgp
4.6 Creating Online Help

It is relatively easy to create online help for user libraries.

An HTML editor and the program “Html Help Workshop” which can be download free of charge from Microsoft via the Internet

http://msdn.microsoft.com/library/tools/htmlhelp/chm/hh1start.htm

It is even easier to create help with tools like “Robohelp” or “FAR”, but they are not free of charge.

4.7 Creating a C Library

The procedure is almost the same as with IEC libraries. The main difference is that the source code for the component is not added directly by double clicking on the function, instead must be added as source files.

Detailed information can be found in the online help. There is a tutorial that explains the exact procedure.

Fig. 11.18: Tutorial for creating a C library
4.8 Reusing Libraries

To reuse a library, the following steps must be carried out.

- **Copy the library from the current project**

  C:\Projekte\ASProg.pgp\Library\“Name of the library“

  This library directory, which exists in all projects, contains all libraries included in the project.

The library directory is divided into five subdirectories:

- **Help:** Online help, if available
- **i386:** Library files for Intel platform (*.br, *.h, *.a File)
- **m68k:** Library files for Motorola platform (*.br, *.h, *.a File)
- **Source:** Source files for the component are stored here. If these source files are deleted, the component can no longer be edited.
- **Temp:** Temporary directory
- **Inserting the library in a standard directory**
  In order to be able to manage libraries correctly, it makes sense to create a separate directory in the AS path.

  e.g.: `C:\BrAutomation\AS\Library\User\Name of the library`

- **Add library directory to LibMan**
  In the new project, the path for the user libraries has to be entered. Now the library can be inserted like standard libraries.

---

**Fig. 11.20: Path settings in LibMan**
5 PG2000 LIBRARIES

There are some differences between Automation Studio and PG2000, therefore instructions are available to simplify conversion from PG2000 to Automation Studio. Automation Studio provides a function which allows these libraries to be imported.

5.1 PG2000 Porting Guide

This tool can be found in the online help for LibMan. The Porting Guide provides detailed instructions for converting PG2000 libraries.
1 OVERVIEW .............................................................................................................. 2

2 ANSI C ................................................................................................................... 3
   2.1 Development History ................................................................. 3
   2.2 Definition of Terms ................................................................. 4
   2.3 Command Groups ................................................................. 6

3 STRUCTURE OF C PROGRAMS ..................................................... 9
   3.1 B&R Expansions ................................................................. 10
   3.2 Creating a C Task ................................................................. 14
   3.3 Variable Declaration ........................................................... 17
   3.4 Data Types ................................................................. 20
   3.5 Line Coverage ................................................................. 22
   3.6 Functions ................................................................. 24
   3.7 Debugger ................................................................. 30
   3.8 Using Arguments with Complex Data Types ..................... 33

4 USING B&R LIBRARIES .......................................................... 36
   4.1 General Information ........................................................... 36
   4.2 Example .............................................................................. 37

5 COMPILER INFO ................................................................................. 41
   5.1 File Types ................................................................. 41
   5.2 Compile Procedure ............................................................. 42
   5.3 GNU C Compiler ............................................................. 44
1 OVERVIEW

B&R offers the right programming language for every application and for every programmers preference. This includes:

- Ladder Diagram (LAD)
- Instruction List (IL)
- Structured Text (ST)
- Sequential Function Chart (SFC)
- B&R Automation Basic (AB)
- ANSI C

LAD Contact, logic and function operations are combined here in a single user interface. Ladder diagram is the simplest form of programming digital and analog processes because of its similarity to a circuit diagram.

IL Instruction list is similar to a machine language. It can be used like LAD for logic operations.

ST This high level language is a clear and powerful programming language for automation systems. Simple standard constructs guarantee fast and efficient programming.

SFC A sequential language that was developed to separate a task into clear units. SFC is well suited for processes where states change in steps, for example: automatic carwash.

AB This B&R high level language is a clear and powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient programming. Previously PL2000

ANSI C This high level language is a powerful programming language for automation systems of the newest generation. Simple standard constructs guarantee fast and efficient application programming.
2 ANSI C

2.1 Development History

C is a programming language which was developed in the 70s together with UNIX at AT&T Bell Laboratories by Dennis M. Ritchie. It uses and expands on the language “B” which was developed by Ken Thompson. This is how “C” came to be. Since then, this programming language has continued its course of success in all areas of programming.

In 1978, Brian Kerninghan and Dennis Ritchie wrote the “K&R White Book” “The C Programming Language”. Kerninghan wrote the main text and Ritchie wrote the technical sections.

Because it is used in many different areas, it became necessary to standardize the language. This was done by the American National Standards Institute ANSI in 1983. Then it became possible to implement the language on different platforms.

2.1.1 Why use a high level language?

Using language constructs eases programming of control tasks and makes a program created with this language much easier to read. Programs can be achieve much higher performance than with normal PLC programming languages.
2.2 Definition of Terms

High Level Language
Common term for programming languages which allow problem oriented formulation and function independent of the computer type where they are running. e.g.: C, PASCAL, ST, AB

C Structured Programming Language.


ANSI C Standardized “C”

Source Code
Consists of programming commands created by the programmer with a text editor and saved in a file. This file contains the source code. This code is compiled and can then be transferred and executed on controllers, PCs etc.

Source Short form of source code.

File / Program / Document
The basic memory unit on a PC. Documents and programs are files. Different data types are possibly assigned different symbols.

Folder A folder can contain files and other folders. To make things clearer, place your work in folders just like you would in your office or at home. Your directories are shown as folders.

Directory / Path
In text oriented operating systems, directory names or paths are used to store files instead of graphic symbols (folders).

Directory Tree
In order to display directories in a clear manner, programs (e.g.: Windows Explorer) show directories as tree structures.
The C editor is a text editor for C source code. The programmer is shown possible entry errors when programming using syntax coloring. This significantly reduces programming times because syntactic programming errors are ruled out.

**Commands**

Each command in C must be terminated with a semicolon.

```
e.g.: res = a + b;
```

**Comment**

Meaningful comments placed in the program make orientation easier for longer source codes. Additionally, the code is clear and easy to follow, even after a long period of time.

Comments are terminated using `/*` and `*/`.

Comments can be written over several lines.

```
e.g.: /* This is a comment.
And this is the second line. */
```
2.3 Command Groups

An overview of the operator types:

- Primary operators
e.g.: Brackets: \( a = (b+c) \times d \);

- Unary operators
e.g.: Negation: \( a = !b \);

- Arithmetic operators

- Shift operators

- Compare operators

- Bit oriented operators

- Logic operators

- Ternary operators
  Operator for conditional expressions

- Assignment operators

- Comma operator
Note

In this section, we will only explain the operators that are meaningful for getting started in C with B&R controllers.

Primary operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Brackets</td>
<td>value = a * b - c; or value = a * (b - c);</td>
</tr>
</tbody>
</table>

Unary operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Unary Negation</td>
<td>a = !b;</td>
</tr>
</tbody>
</table>

Arithmetic operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Assignment</td>
<td>a = b;</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>a = b + c;</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>a = b - c;</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>a = b * c;</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>a = b / c;</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (remainder of division)</td>
<td>a = b % c;</td>
</tr>
</tbody>
</table>
Compare operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>if (a &lt; b)</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>if (a &gt; b)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>if (a &lt;= b)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>if (a &gt;= b)</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>if (a == b)</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>if (a != b)</td>
</tr>
</tbody>
</table>

Bit oriented operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>And - bit mode</td>
<td>a = b &amp; c;</td>
</tr>
<tr>
<td></td>
<td>Or - bit mode</td>
<td>a = b</td>
</tr>
<tr>
<td>^</td>
<td>Or - exclusive</td>
<td>a = b ^ c;</td>
</tr>
</tbody>
</table>

Logic operators

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>And</td>
<td>if (a &gt; 0) &amp;&amp; (b &gt; 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 STRUCTURE OF C PROGRAMS

Like any other task, a C task can consist of several functions.

Now we will show the individual possibilities using an example:

The following C code increases a counter variable each time the function with the name “CyclicFunction(..)” is called, or is set to a start value by calling the function “Initialization(..)” and is set then to an end value when starting the function “Terminate(..)”.

```c
INT variable; /* Declaration of the variables */
void Initialization(void) {
    variable = 1; /* Assign start value */
}
void CyclicFunction(void) {
    variable = variable + 1; /* Value increased by 1 */
}
void Terminate(void) {
    variable = 0; /* Reset value */
}
```

Code blocks are represented in C using curved brackets.

```c
{ Start code block
    /* Code block */
} End code block
```
3.1 B&R Expansions

3.1.1 Include Files

In order to use certain possibilities on the PCC, we will first include the header file: “plc.h” which contains definitions for all necessary macros using the following instruction:

```c
#include <bur\plc.h>
```

Like all other B&R system header files, the header file “plc.h“ is found in the AS install directory:

```
BrAutomation\AS\GnuInst\m68k-elf\include\bur\plc.h
```

and can be added to the C task by marking the C task name and pressing the [ENTER] key, or using the main menu item **Insert: File**. This is not absolutely necessary, but has the advantage of continually monitoring the header file for changes.
3.1.2 Analyzing the Entries in the Header File “plc.h”

- Definition of the Variables:

<table>
<thead>
<tr>
<th>Macro name/Attribute</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>_GLOBAL</td>
<td>Code for a PCC global variable.</td>
</tr>
<tr>
<td>_LOCAL</td>
<td>Code for a task local variable.</td>
</tr>
</tbody>
</table>

- Definition of the Functions:

<table>
<thead>
<tr>
<th>Macro name/Attribute</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>_INIT</td>
<td>Code for an INIT routine. Only executed when booting the PCC with a cold start, warm start, or when transferring a task.</td>
</tr>
<tr>
<td>_CYCLIC</td>
<td>Code for the function which is called cyclically. The function name <strong>cannot be “main”</strong></td>
</tr>
<tr>
<td>_EXIT</td>
<td>Code for the EXIT routine. Called once when deinstalling the task. Also executed when the task is transferred again.</td>
</tr>
<tr>
<td>_NONCYCLIC</td>
<td>Code for a function which is executed in the system idle time. (only for special applications)</td>
</tr>
</tbody>
</table>
Using this macro in our C program results in the following code:

```c
#include "plc.h"    /** B&R-Standard files **/

_GLOBAL INT variable;  /** PCC-global variables **/

/** This function is only used when booting the PCC, or when downloading the task **/
_INIT void Initialization(void)
{
    variable = 1;   /** Assign start value **/
}

/** This function is executed cyclically **/
_CYCLIC void CyclicFunction(void)
{
    variable = variable + 1;  /** Value increased by 1 **/
}

/** This function is only called when deinstalling the task **/
_EXIT void Terminate(void)
{
    variable = 0;  /** Reset value **/
}
```
Example

Create a new project and test the previous example using line coverage.

Project Name: c_proj
Task Name: c_task1
Resource: C#3
3.2 Creating a C Task

C tasks are inserted in the project just like LAD tasks.

This type of task can consist of several files, therefore these source files are added under the task symbol using “Insert File”.

Fig. 12.3: C task, Insert File dialog box
In order to manage the project in a clearer manner, all source files are grouped together in a “Source” directory. A new folder can be created by simply clicking on the marked symbol.

![Fig. 12.4: Create new folder](image)

Then the file name is entered.

**Note**

The extension “.c” also has to be entered!
Double-clicking on the file opens the editor.

Fig. 12.6: Opening the C editor
3.3 Variable Declaration

In addition to the macros for "_GLOBAL" or "_LOCAL" shown above, the variable declaration in the C file also provides the possibility to use C global or C local variables.

The difference between PCC PVs and C variables is the target memory. PCC PVs which are entered in the AS database, Main menu item **Open: Declaration**, are stored in Dual Ported Ram. C variables are defined in User RAM (freely available).

<table>
<thead>
<tr>
<th>DPR PVs: _GLOBAL / _LOCAL</th>
<th>C Variables: C global / C local</th>
</tr>
</thead>
<tbody>
<tr>
<td>The variable values are remanent</td>
<td>C variable are initialized with 0 when switching on the controller.</td>
</tr>
<tr>
<td>The max. size of an analog PV is limited to approx. 30kByte.</td>
<td>The size of a C variable only depends on the memory available.</td>
</tr>
<tr>
<td>_GLOBAL PVs can be assigned to the hardware.</td>
<td>C variables cannot be assigned to the hardware.</td>
</tr>
<tr>
<td>_GLOBAL, _LOCAL PVs can be displayed (e.g.: using PVI, NET2000, etc.).</td>
<td>C variables cannot be referenced via PVI, NET2000, etc. That means: They are not visible in the WATCH window.</td>
</tr>
<tr>
<td>Initialization of PVs in the variable declaration or in the INIT routine.</td>
<td>Initialization possible directly during definition in the C file (e.g.: int variable = 123;).</td>
</tr>
<tr>
<td>No multidimensional arrays (matrix).</td>
<td>Multidimensional arrays are allowed</td>
</tr>
<tr>
<td>Structures have a max. of 16 layers.</td>
<td>No limits for structures.</td>
</tr>
<tr>
<td>Enumerations data types are not available.</td>
<td>Enumeration data types are supported.</td>
</tr>
<tr>
<td>When using pointers, only the address is entered as UDINT in variables / structures.</td>
<td>In C, pointers can be used as normal.¹</td>
</tr>
</tbody>
</table>

¹ With the exception of PCC variables declared with _LOCAL (first reference is a dynamic variable).
3.3.1 Scope of the Variables

- A PV defined with _LOCAL is local in the task and global for the task files.
- A PV defined with _GLOBAL is PCC global and also global for the task files.
- A variable declared as C global is global for all C files and its scope is limited within a task.
- A local C variable is only valid in the function where it was defined.

In order to use a global C variable in a tasks in several files, it must be declared in a file without the attribute “external”. In this way, the memory for this variable is reserved. In all other files where this variable is accessed, it have the attribute "external". The attribute external indicates to the compiler that the variable is already declared in another file.
IMPORTANT

- For each PCC task, **one function must** have the attribute "CYCLIC". But **only one function** can have this attribute.
- For each PCC task, a function can have the attributes "_INIT", "_EXIT" if required.
- DPR PVs have the attribute "_GLOBAL" or "_LOCAL". In this way, these variables are managed by AS in the project variable declaration **Open: Declaration**.
- If one or more DPR PVs are no longer used in the programs, they remain in the project and reserve memory in the DPR until a “Build All” **Project: BuildAll**.
- DPR PVs are referenced using a max. of 32 characters. C variables can be longer.
- For the variable types, make sure that C does not have any rules regarding bit length for an integer variable (int). Therefore this length depends on the processor and is 32 bit for the B&R PCC.

In order to work with IEC1131 data types in C, the user is provided the B&R header file named: “plctypes.h”.

**Directory for plctypes.h:**

`BrAutomation\AS\Gnuinst\m68k-elf\include\bur\plctypes.h`
### 3.4 Data Types

Various data types are defined in IEC61131-3. Some of these data types do not have a corresponding type in the ANSI C standard. Therefore B&R Automation Studio has declared some data types in the file “plctypes.h” so that all IEC1131 data types can be used in C.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>IEC1131-3</th>
<th>B&amp;R C</th>
<th>ANSI C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit</td>
<td>BOOL</td>
<td>plcbit¹</td>
<td>unsigned char</td>
</tr>
<tr>
<td>8 bit with sign</td>
<td>SINT</td>
<td>signed char</td>
<td>signed char</td>
</tr>
<tr>
<td>8 bit without sign</td>
<td>USINT</td>
<td>unsigned char</td>
<td>unsigned char</td>
</tr>
<tr>
<td>16 bit with sign</td>
<td>INT</td>
<td>short</td>
<td>short</td>
</tr>
<tr>
<td>16 bit without sign</td>
<td>UINT</td>
<td>unsigned short</td>
<td>unsigned short</td>
</tr>
<tr>
<td>32 bit with sign</td>
<td>DINT</td>
<td>long</td>
<td>long</td>
</tr>
<tr>
<td>32 bit without sign</td>
<td>UDINT</td>
<td>unsigned long</td>
<td>unsigned long</td>
</tr>
<tr>
<td>32 bit time difference in milliseconds</td>
<td>TIME</td>
<td>plctime¹</td>
<td>signed long</td>
</tr>
<tr>
<td>32 bit date in seconds since 1970²</td>
<td>DT</td>
<td>plcdt¹</td>
<td>unsigned long</td>
</tr>
<tr>
<td>Zero terminated string with length x³</td>
<td>STRING(x)</td>
<td>plcstring¹ [x+1]</td>
<td>char [x+1]</td>
</tr>
<tr>
<td>Floating point representation 32 bit</td>
<td>REAL</td>
<td>float</td>
<td>float</td>
</tr>
</tbody>
</table>

**IMPORTANT**

In order for the compiler to view plcbit variables as unsigned char, the user must make sure that only 0 or 1 is assigned.

¹ Defined by B&R in plc.h
² Unix standard format for date entries
³ In IEC1131-3, the final 0 byte is not counted, it is in ANSI C
Example

Create a solution to the following task in C.

Check “TempAct”. If “TempAct” is smaller than “TempSet”, then bit variable „heating“ should be set to 1. Otherwise “heating” should be cleared.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scope</th>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempAct</td>
<td>INT</td>
<td>global</td>
<td>IP5.0.5.1</td>
<td>----------</td>
<td>Analog IN, Chan. 1</td>
</tr>
<tr>
<td>TempSet</td>
<td>INT</td>
<td>global</td>
<td>IP5.0.5.2</td>
<td>----------</td>
<td>Analog IN, Chan. 2</td>
</tr>
<tr>
<td>heating</td>
<td>BOOL</td>
<td>global</td>
<td>QP5.0.4.11</td>
<td>* remnant</td>
<td>Dig. OUT, Chan. 11</td>
</tr>
</tbody>
</table>

Project Name: c_proj

Task Name: c_task2

Resource: C#3
3.5 Line Coverage

View: Monitor or <Ctrl><M>

AS is first switched to Monitor Mode.

Object: Start Line Coverage

Start -> by clicking on the “green” lamp symbol

With Line Coverage, the values are taken from the controller and shown in the respective program context.

The arrows on the left indicate that a line is being processed. Figure 11.7 the line „heating = 0“ is not currently being processed.

If the mouse cursor is placed over a variable, then a „Tool Tip“ is opened which shows the current value of the variable on the controller.

![Fig. 12.7: Line Coverage](image-url)
Example

Test “c_task2” using Line Coverage.
3.6 Functions

3.6.1 General Information

Sub-programs which are referred to as functions are a major part of C. They are used to structure the program in a clearer manner. These sub-programs are very similar to function blocks, but can only be used by C tasks.

3.6.2 Structure of a Function

<table>
<thead>
<tr>
<th>Return value</th>
<th>Function name</th>
<th>Arguments</th>
</tr>
</thead>
</table>

e.g.:

```c
void CyclicFunction(void)
{
    //...
}
```

- Returned value
  Provides the function the possibility to return a value to the function where it was called. In our example, we don’t have a return value, therefore `void` (void = empty).

- Functions name
  Name used to call the function in the program.

- Arguments
  Values given to the function. In our example, no values are provided, therefore: `void`
3.6.3 Providing an Arguments as a Value

Program sections which are often reused should be placed in sub-programs (=functions). The arguments provided are processed in these functions. If necessary, the result can be given back to the function that made the call as a return value.

For example, the operating system provides the Init function the current boot information.

```c
#include <bur\plc.h> /* B&R Standard files */
#include <bur\plctypes.h> /* B&R Standard files */

_LOCAL DINT Boot information;

/* This function is only used when booting the PCC,
* or when downloading the task */

_INIT void Initialization (DINT PCC_info)
{
    Boot information = PCC_info;
}
```

Argument: (DINT PCC_info)

The argument is saved in a local C variable in the function and can only be used by this function.

To provide several arguments for a function, they must be separated by commas.
### 3.6.4 File Local Function

We want to add two numbers with a function and save the result in a "_LOCAL" variable.

```c
#include <bur\plc.h>  /* B&R Standard files */
#include <bur\plctypes.h>  /* B&R Standard files */

LOCAL DINT Boot information;
LOCAL DINT Result;
LOCAL INT Value1,
Value2;

/* Declaration of the function prototypes */
DINT Add(INT Value1, INT Value2);

/* This function is only used when booting the PCC, or when downloading the task */
_INIT void Initialization (DINT PCC_info)
{
    Boot information = PCC_info;
}

_CYCLIC void Cyclic(void)
{
    /* Call the function */
    Result = Add(Value1, Value2);
}

/* the main function */
DINT Add(INT Value1, INT Value2)
{
    DINT Res;  /* Definition of a local C variable */
    Res = Value1 + Value2;  /* Calculating the sum */
    return( Res );  /* Return the result to the function that made the call */
}
```
Example

Create a local function, which multiplies 3 values. Test the function in a task.

Task Name: calc
Resource: C#1
3.6.5 Task Global Function

All functions in a task can access this sub-program (File 2).

**Task_1:**

**C-File1:**
```c
#include <bur\plc.h>       /* B&R Standard files */
#include <bur\plctypes.h>  /* B&R Standard files */

_LOCAL DINT Result;
_LOCAL INT Value1, Value2;

/* Declaration of the function prototypes */
external DINT Add(INT Value1, INT Value2);

_CYCLIC void CYCLIC(void)
{
    /* Call the function */
    Result = Add(Value1, Value2);
}
```

**C-File2:**
```c
#include "plc.h"           /* B&R Standard files */
#include "plctypes.h"      /* B&R Standard files */

DINT Add(INT Value1, INT Value2)
{
    DINT Res;  /* Definition of a C local variable */
    Res = Value1 + Value2;  /* Calculation */
    return( Res );  /* Return value */
}
```

In C File1, sub-program Add()) is called in function Cyclic(). The **function code** is contained in C File2. Therefore the compiler in C File1 must be informed that the code for Add() is **external**.
Example

Change task “calc” so that the addition is coded in a separate file.

Task Name: calc
Resource: C#1
3.7 Debugger

3.7.1 Set / Clear Stop Point

With this function, break points are set on the controller **which stop the system.**

The Debug output window on the lower edge of the AS window shows Debugger information. The toolbar for the Debugger is opened.

The program can be tested in single step mode with the “single step” or “procedure step” function.

“Next” causes processing to continue to the next stop point or cycle.

3.7.2 Single Step

All instruction lines are executed. If functions are called, they are executed if the source code is available.

3.7.3 Procedure Step

Functions are completely executed. Then the program counter Stopps after the function call.

---

Fig. 12.8: C Task Debugger
3.7.4 Debugger: Watch Fenster

This window is opened if the program on the controller gets to a breakpoint. If a variable in the source window is marked and pulled into the Watch window (Drag & Drop), then the value of the variable, the array or the structure is shown.

![C Task Debugger Watch Window](image)
Example

Test the Debugger using an example.
3.8 Using Arguments with Complex Data Types

3.8.1 Application Example 1

Creating a slave pointer. (Slave pointer => A function that determines the maximum using a current value).

Note

If a function should return more than one value to the function that made the call, or if a function should work directly with the memory for the function that made the call, this cannot take place using the function return().

```c
#include <bur\plc.h>  /** B&R Standard files           **/
#include <bur\plctypes.h>  /** B&R Standard files          **/
/*------------------------------------------------------------*/
/** Constant definition                                    **/
/*------------------------------------------------------------*/
#define TRUE 1
#define FALSE 0
/*------------------------------------------------------------*/
/** TYPE DEFINITION FOR MEASSTRUCTURE                      **/
/**-------------------------------------------------------------*/
/** init: Used to set max_value to 0.                      **/
/** current_value: Current value to be evaluated.         **/
/** max_value: Maximum value of current_value             **/
struct MEASSTRUCTURE {
    BOOL init;
    INT  current_value;
    INT  max_value;
};
/*-------------------------------------------------------------*/
/** VARIABLE DECLARATION                                    **/
/*-------------------------------------------------------------*/
/** Variable given to the slave pointer                    **/
_LOCAL struct MEASSTRUCTURE fillstatus;
/*-------------------------------------------------------------*/
/** FUNCTION PROTOTYPES                                    **/
/**-------------------------------------------------------------*/
/** The function slavepointer() does not make a copy of the variable given, instead works directly with the variable for the function that made the call, therefore a star must be inserted before “data” in the function header          **/
void slavepointer(struct MEASSTRUCTURE *data);
```
/**-------------------------------------------------------------**/
/** INITIALIZATION OF THE TASKS (INIT-SP)                        **/
/**-------------------------------------------------------------**/
_INIT  void initup(DINT OS_info)
{
    fillstatus.init = 1;
    slavepointer(&fillstatus);
}

/**-------------------------------------------------------------**/
/** CYCLIC SECTION OF TASK                                      **/
/**-------------------------------------------------------------**/
_CYCLIC void cyclic_func(void)
{
    /** Maximum evaluation using slave pointer                 **/
    slavepointer(&fillstatus);
}

/**-------------------------------------------------------------**/
/** If the current value is > the maximum value,               **/
/** the current value is set to the maximum value.             **/
/**-------------------------------------------------------------**/
/** void slavepointer(struct MEASSTRUCTURE *data)               **/
/**-------------------------------------------------------------**/
/** The function slavepointer() does not make a copy of the    **/
/** variable given, instead works directly with the variable   **/
/** for the function that made the call, therefore a star must **/
/** be inserted before "data" in the function header           **/

void slavepointer(struct MEASSTRUCTURE *data)
{
    if (data->init == TRUE)
    {
        data->max_value = 0;
        data->init = 0;
    } /** end if (data->init .. **/

    if(data->current_value > data->max_value)
    {
        data->max_value = data->current_value;
    } /** end if(data->current_value > data->max_value) **/

    /** return without return value **/
}

Note
Detailed information concerning working with pointers can be found in the respective C literature.
3.8.2 Application Example 2

Determine the length of a strings in a function.

```c
#include <bur\plc.h>
#include <bur\plctypes.h>

/** Constant definition **/
#define TRUE 1
#define FALSE 0
#define END_OF_TEXT 0 /** Strings are zero terminated **/

/** VARIABLE DECLARATION **/
_LOCAL STRING    text[20];
_LOCAL INT       length;

/** FUNCTION PROTOTYPES **/
INT string_len(STRING string[]);

/** INITIALIZATION OF THE TASKS (INIT-SP) **/
_INIT    void init(void)
{    
    text[0]= END_OF_TEXT;
}

/** CYCLIC SECTION OF TASK **/
_CYCLIC void cyclic_func(void)
{    
    length = string_len( &(text[0]) );
}

/** C strings are zero terminated, therefore the loop searches for
the first 0. Then the length is known. **/
INT string_len( STRING string[])
{    
    INT len = 0;

    while (string[len] != END_OF_TEXT)
    { len++;
    }
    return(len);
}
```
4 USING B&R LIBRARIES

4.1 General Information

Libraries or standard functions are used to efficiently create programs and offer a simple, complete solution for often required functions.

The following components are needed to use B&R standard functions in a C task:

- Library Header File e.g.: “standard.h”
  Contains the variable type declaration and the function type declaration.
- Library Archive File e.g.: „standard.a“
  Contains the link information for the start address in the library code of the system files

The files for various libraries are found directly under the library path for the project. Libraries imported in the project are copied by LibMan directly into this directory.
4.2 Example

4.2.1 Inserting the Library in the Project

A standard function from the STANDARD library should be used, and the following settings must be made in the project.

- Insert the desired library in the Library Manager. Inserting the library according to the set operating system version copies the necessary files to the path:

  e.g.: C:\projects\prj_name.pgp\Library

- Inserting the library header files and the archive files

**IMPORTANT**

The library *archive file* must be placed in the tree structure after the C file that accesses a corresponding library function.
4.2.2 Inserting the Header File

Only the library header file: “standard.h” has to be included in the C code. The library archive file “libstandard.a” is only added to the task using **Insert: File**.

If e.g.: the function TON(..), a turn on delay function from the standard library, should be used, this results in the following program:

```c
#include <bur\plc.h>
#include <bur\plctypes.h>
#include <standard.h>   /** Library Header File **/

_LOCAL TON_10mstyp  ton_para;  /** TON parameter variable **/

_GRAPHIC BOOL  gDiStart,   /** start TimerOnDelay **/
   gDoRelay;   /** Output **/

_CYCLIC void use_standard_func(void)
{
    ton_para.PT = 100;   /** set preset-time to lsec. **/
    ton_para.IN = gDiStart;

    TON_10ms(&ton_para);

    gDoRelay = ton_para.Q;   /** set output **/
}
```
Example

Use the B&R standard library and the function block TOF
4.2.3 Using C Standard Libraries

This example shows the use of the mathematics library. Monitor the progress of “x” and “y” with the Tracer.

```c
#include <bur\plc.h>
#include <bur\plctypes.h>
#include "math.h"

/**-------------------------------------------------------------**/
/** Constant definition **/
/**-------------------------------------------------------------**/

#define TRUE 1
#define FALSE 0
#define SCANTIME 0.01      /** Task class cycle time **/

/**-------------------------------------------------------------**/
/** VARIABLE DECLARATION **/
/**-------------------------------------------------------------**/

_LOCAL REAL x, y, freq, t;

/**-------------------------------------------------------------**/
/** INITIALIZATION OF THE TASK (INIT-SP) **/
/**-------------------------------------------------------------**/

_INIT void init(void)
{
    freq = 1.0;
}

/**-------------------------------------------------------------**/
/** CYCLIC SECTION OF THE TASK **/
/**-------------------------------------------------------------**/

_CYCLIC void cyclic_func(void)
{
    t = t + SCANTIME;
    x = sin( M_TWOPI * freq * t );
    y = cos( M_TWOPI * freq * t );
}
```
5 COMPILER INFO

5.1 File Types

*.c  C Source Files
    Definition of the variables. Implementation of the functions

*.h  C Header Files
    Prototypes of functions and declaration of variables that should be used for several C source codes. These files are included in the C source code using the preprocessor instruction #include.

*.s  Assembler source code text.

*.o  Object Files:
    Compiled source file.

*.a  Library Files
    Archive made up of several compiled source files. Often simply called library. Take note that archive files always have to be included at the end of the task.
5.2 Compile Procedure

- **Preprocessor**
  The preprocessor includes the header files in the source code and replaces the macros. Macros are special instructions for the preprocessor. The `#include` instruction is also a preprocessor instruction.

- **Parser**
  The parser is a program element which divides the source code into individual sections, so that it can be processed by other areas of the compiler. The parser is also often used to check syntax. That means, the amount and type of parameters given to the functions.

- **Compiler**
  The compiler is a program that converts the source code to an assembler file (*.s). At the beginning of the compile procedure, the source code is checked by the parser.
• **Assembler**
  The assembler creates machine code with linker information from the assembler code. The Object File (*.o). For most systems, this file must be processed further e.g. to group several object files or to correctly call functions contained in libraries. This is done by the Linker.

• **Linker**
  The Linker creates a single code file from the various object files and libraries, the “task.out” file.

• **Backend, Taskbuilder:**
  The output file from the Linker can be converted to a “task.br” file with AS database information and some hardware specific modifications. This file can then be processed by the PCC.
5.3 GNU C Compiler

We use a ported GNU C compiler as C program compiler. GNU originates from the UNIX world and has proven to be a reliable code generator that is used worldwide.

To be able to use the many possibilities offered by this compiler, documentation is provided in universal HTML format in the directory BrAutomation\AS\Gnudoc\index.htm.

Default values for general use are defined in AS. The user only has to make additional settings if required.

Changes can be made by positioning the marking cursor on the task and selecting the main menu item **Edit: Properties**. The properties menu can also be reached using the right mouse button when the task is selected.

![Fig. 12.15: C task settings](image)

Compiler Option:

- `-O0` Code optimization off (useful for Debugging)
- `-Dmacro` Define Macro (conditional code generation)
SEMINAR REVIEW

1 SEMINAR REVIEW .................................................................................. 2
2 SEMINAR OVERVIEW ........................................................................ 3
3 SALES LOCATIONS .............................................................................. 4
1 SEMINAR REVIEW

- B&R Automation Studio
- B&R Automation Runtime
- B&R Automation Target
- B&R Automation Net
- Project Guidelines
- Sequential Function Chart
- AB Automation Basic
- Data Handling
- TPU Code Linker
- Library Manager Introduction
- ANSI C
2 SEMINAR OVERVIEW

B&R AUTOMATION SOFTWARE

B&R CONTROL SYSTEMS

B&R MOTION SYSTEMS

B&R PANEL SYSTEMS
3 SALES LOCATIONS

The most current addresses and product information can be found at:

HTTP://WWW.BR-AUTOMATION.COM