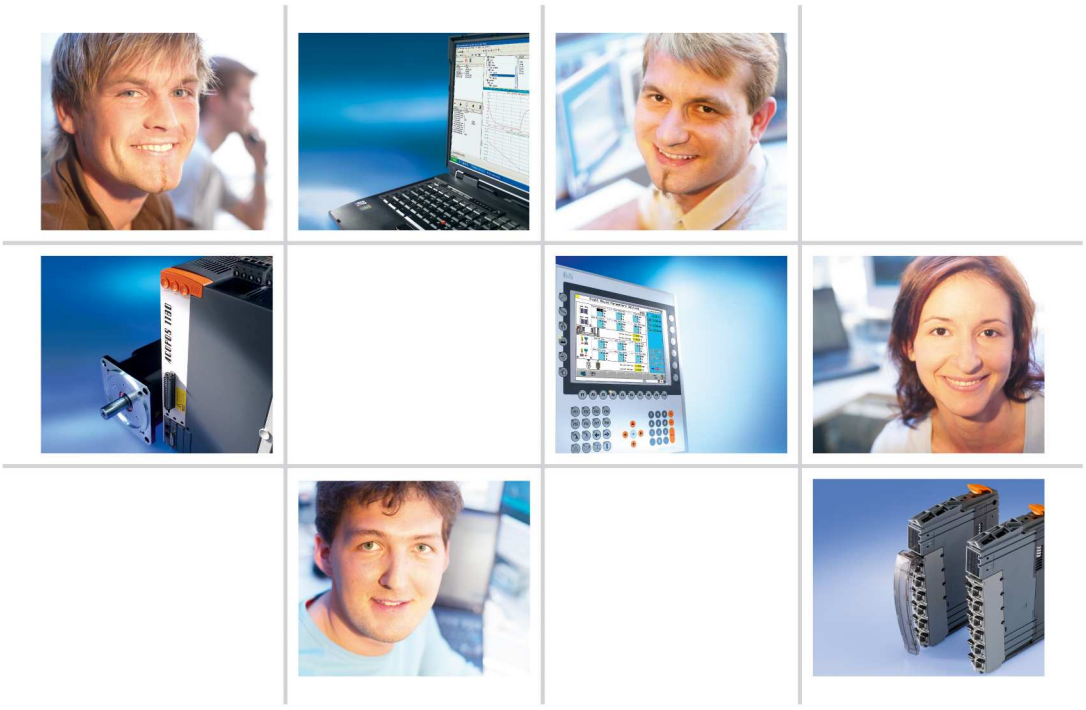
Automation Studio Libraries I
TM260



Perfection in Automation

www.br-automation.com

Prerequisites

Training modules:	TM213 – Automation Runtime
	TM240 – Structured Text (ST)
Software:	Automation Studio 2.5
	Automation Runtime 2.90
Hardware:	None

Table of contents

1. INTRODUCTION

A library is generally understood as an establishment, where knowledge and information is organized into logical and thematic categories. This knowledge should be accessible at any time and easily available for interested visitors.

When creating software, **program libraries** are used to collect program functions for tasks that belong together.
Libraries are not independent running programs. They are program organization units. Libraries contain completed functions and function blocks, which can be used by different programs. This prevents having to re-develop routines that are complex or often used.

As you can see, libraries are an important aid for effective and structured software development.

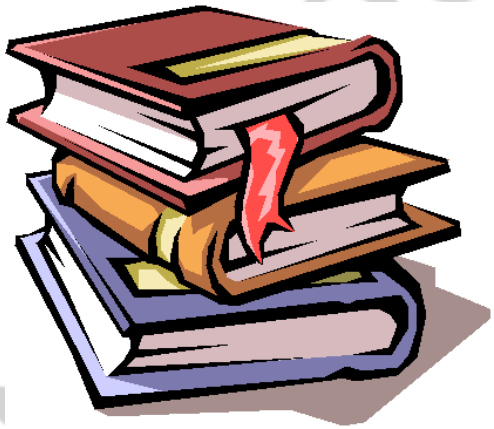


Fig. 1 Books

In this training module, we will explain exactly how to work with libraries, and the functions and function blocks contained in libraries when creating application software with Automation Studio.
To start off, we will take a closer look at the features and advantages as well as the integration and management of libraries in Automation Studio. Participants will also get an overview of the range of B&R standard library functions and will be shown the possibilities for creating user-specific libraries and functions / function blocks.

Examples and exercise tasks will complement the basic theory and shed some light on practical usage.

1.1 Objectives

Participants will get to know the function principle and the advantages of libraries.

Participants will get an overview of the multiple functionalities of B&R's standard libraries.

Participants will be able to locate information in the Automation Studio online help for using and configuring the functions and function blocks.

Participants will learn how to work with and how to effectively apply the libraries and functions / function blocks.

By the end of this training module, participants will be able to create a library and corresponding functions / function blocks to meet their individual demands.

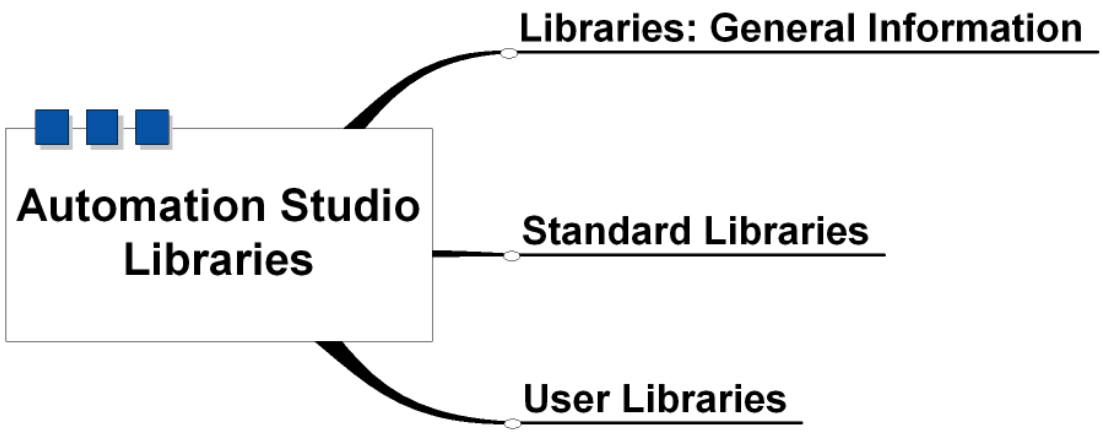


Fig. 2 Overview

2. LIBRARIES: GENERAL INFORMATION

In the IEC, the term **library** is not specifically defined in regard to functions or function blocks. However, libraries have proven to be more than useful for organizing **pre-defined standard functions**. Functions or function blocks for a specific area of application are grouped respectively in a library and can then be executed in the control program, as long as the library is integrated in the project.

However, a library also includes **data types** and **constants**, which are used internally by the **functions** and **function blocks** or for setting parameters.

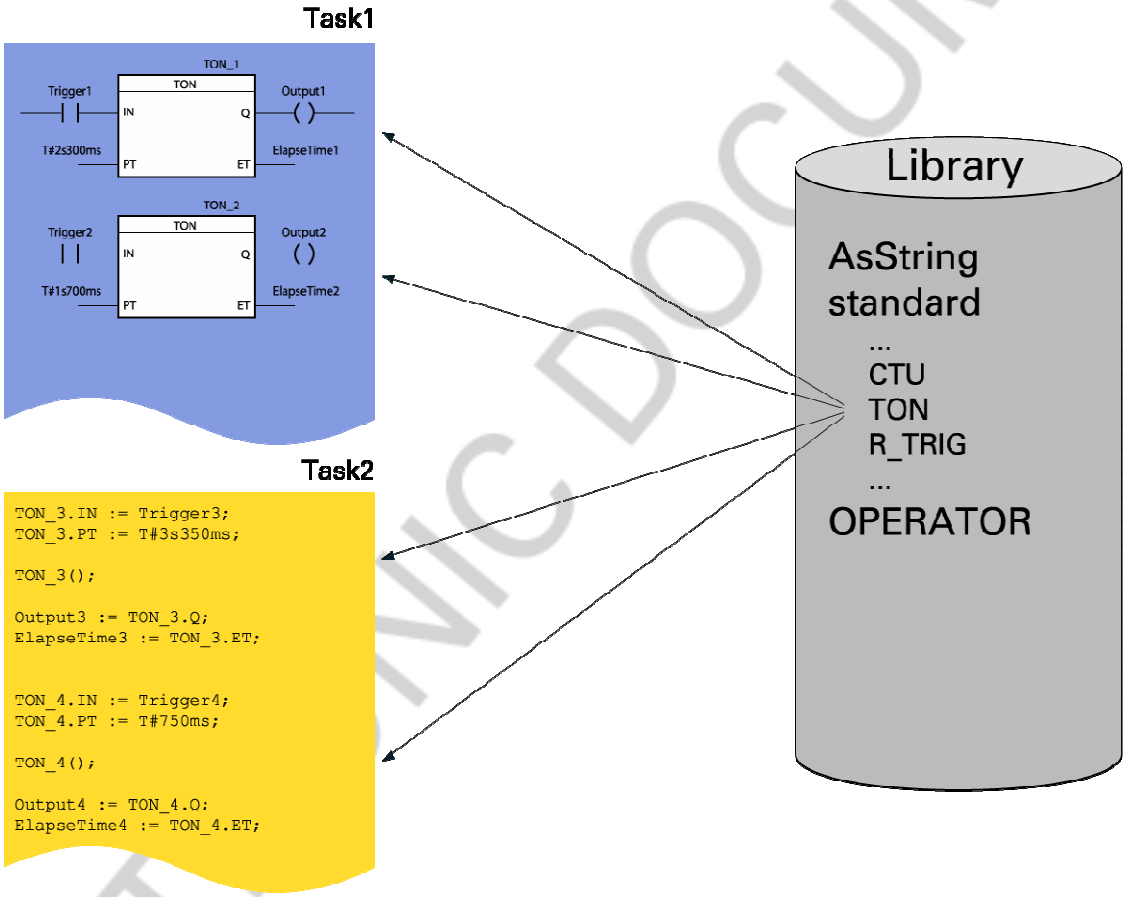


Fig. 3 Using a function block from a library for multiple tasks

Among many other benefits, this method of organizing software functions provides the following advantages:

- Functions and function blocks from a specific area of application are **grouped together in one location** and can be managed and executed from there.
- A library can easily be passed on as a **complete unit** which allows it to be used in all other projects.
- Using functions and function blocks **makes it easier to exchange** the software, because these program sections can be tested separately or can be accepted as is (already tested for functionality).
- Because of their unique system behavior, functions and function blocks are used to **standardize** program code.
- Unlike tasks, functions and function blocks have a **uniquely defined interface**, which is easy to document.
- The functions and function blocks in a library are **saved just one time** respectively on the controller (i.e. additional memory is not used when the function or function block is executed multiple times).

2.1 Functions and function blocks

The program functionalities grouped in libraries are divided into two types:

- Functions
- Function blocks

These two types differ in behavior and how they are used.

2.1.1 Definition of "function"

A function is a program organizational unit which returns exactly one value. Therefore, it has **just one output**, but can have **any number of inputs**. Unlike function blocks, functions **do not have any static memory**. With only a few exceptions (e.g. time and IO – read functions), this means that it always returns the same output value when called repeatedly with the same input parameters.

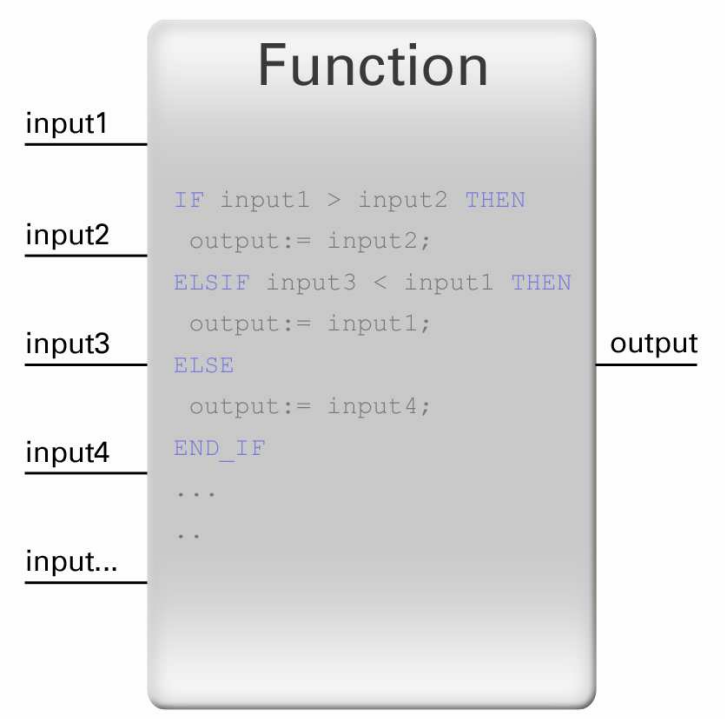


Fig. 4 Function

2.1.2 Definition of "function block"

A function block is a program organizational unit which can return one or more values. Therefore, it can have **one or more inputs and outputs**.

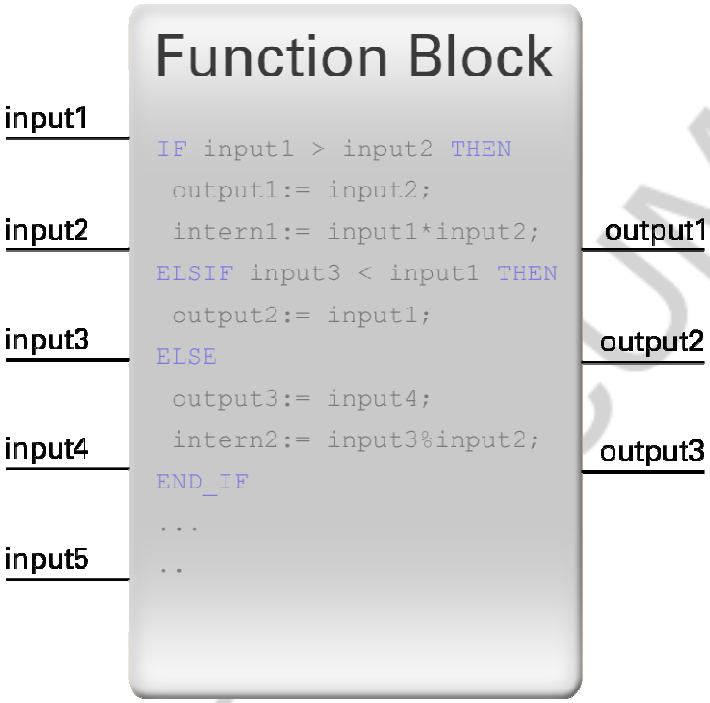


Fig. 5 Function block

An **instance** of a function block must be created before it can be used. This is essentially a data structure, which contains all of the parameters that the function block uses (i.e. inputs, outputs, and internal variables).
By using a data structure, function blocks have **a static memory**. When called repeatedly with the same input parameters, the output values can also change.

In some cases, function blocks, which require a great deal of system resources or access hardware, might have to be called **repeatedly using multiple cycles**. This makes it possible to wait for a response from the hardware and can reduce the load that the function block puts on the system.

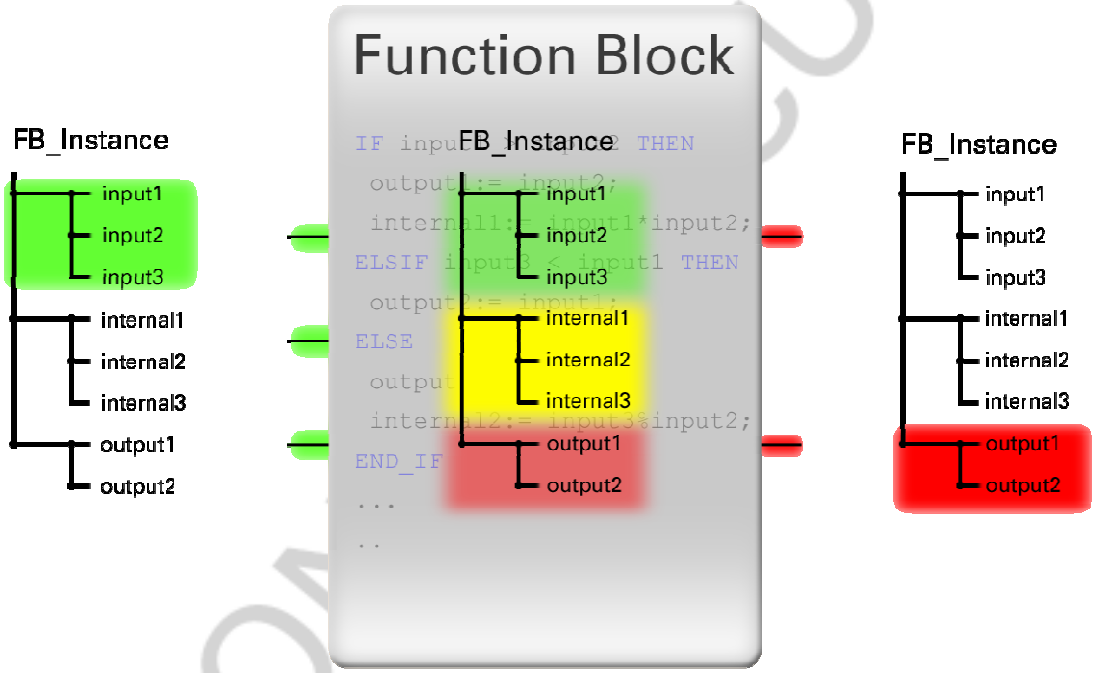


Fig. 6 Function block with instance structure

2.2 Library Manager

The Library Manager is the interface used to completely **manage the libraries** used in a project. This includes managing standard libraries and libraries from third-party suppliers as well as offering support when creating user libraries.

The Library Manager is fully integrated in Automation Studio and can be accessed via the menu option **Open:Library Manager**.

The Library Manager is divided into two main sections:

The libraries integrated in the project are displayed in the left window with the corresponding functions and function blocks.

Information and properties of the element selected in the left window are displayed on different tabs in the right window.
This includes:

- Data types and constants used by the library.
- Additional dependencies from other libraries.
- Parameter declaration of the functions and function blocks.
- Management of the source code files for ANSI C libraries.

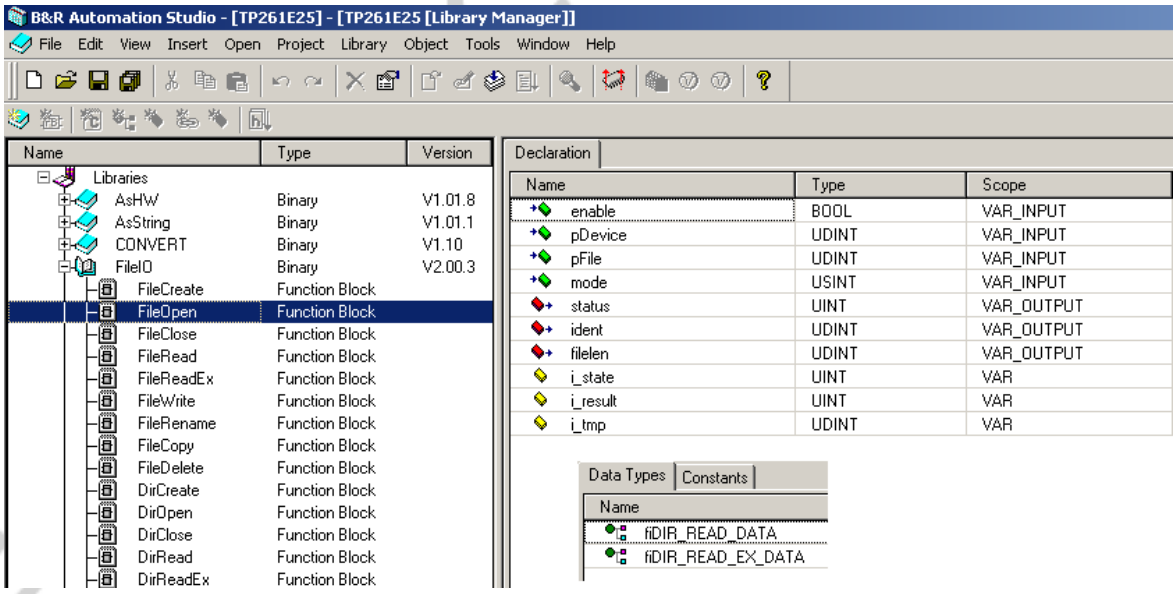


Fig. 7 Library Manager – Declaration of the "FileOpen" function block

2.2.1 Declaration of the functions and function blocks

When choosing a function or function block, all inputs, outputs and internally used variables are displayed in the declaration with their data types.

2.2.2 Data types and constants

The data types and constants used by the library are automatically added to the project when the library is integrated and are globally available.

The name of the library is entered in the data type editor as owner for data types that are unique to the respective library.

2.2.3 Integrating new libraries

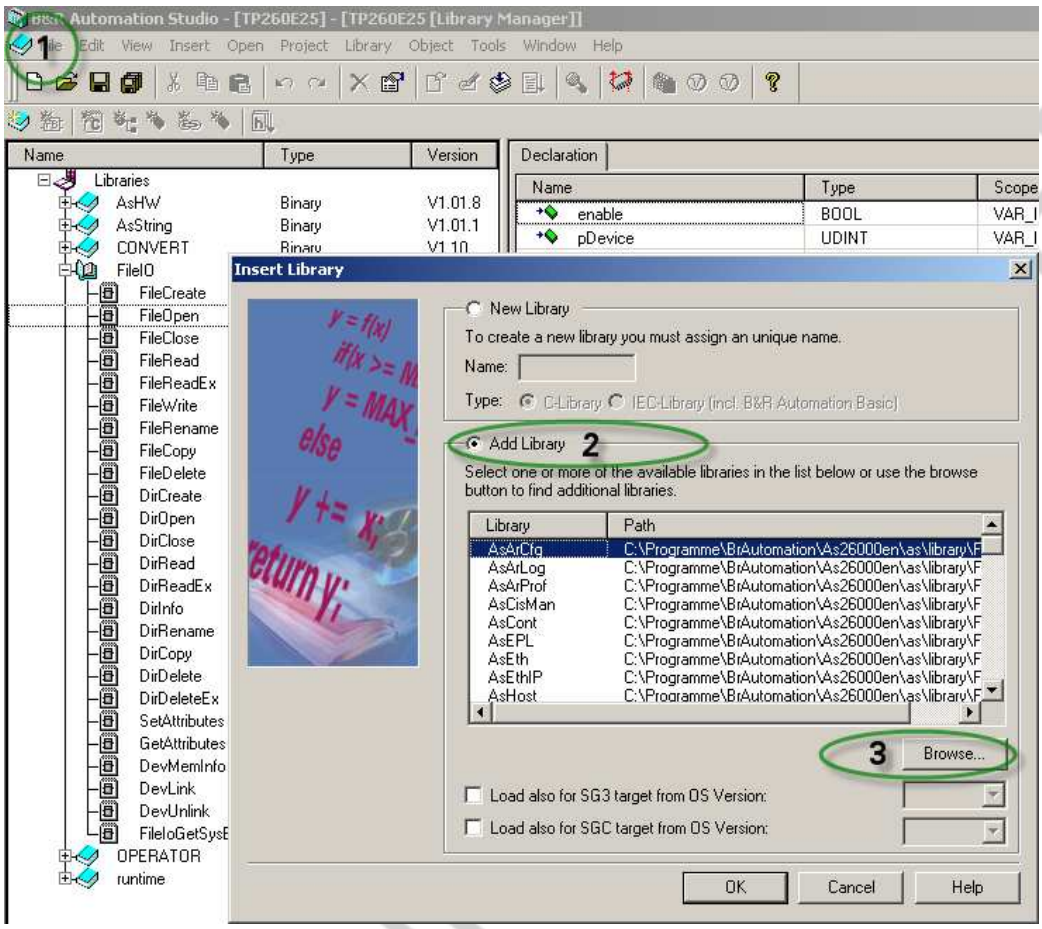


Fig. 8 Adding new libraries

1 A wizard is provided for integrating new libraries into the project. This wizard is opened using the "Insert Library" button.

2 Select "Add Library" if a standard library is required from the current Automation Runtime version. The desired library can then be selected from the list.

3 The "Browse" button can be used to add libraries that are not stored in the standard directory (e.g. user libraries).

2.2.4 Storing libraries

The B&R standard libraries are stored in the Automation Studio installation directory (e.g. "BR_AS_250_L001") in a subdirectory with the **name of the corresponding Automation Runtime version**.

The integrated libraries are also stored in a separate directory within the project directory (e.g. "TM2xx.pgp").

If a function or a function block from a standard library is used in the project, then the entire library is copied from the Automation Studio installation directory to the project directory.

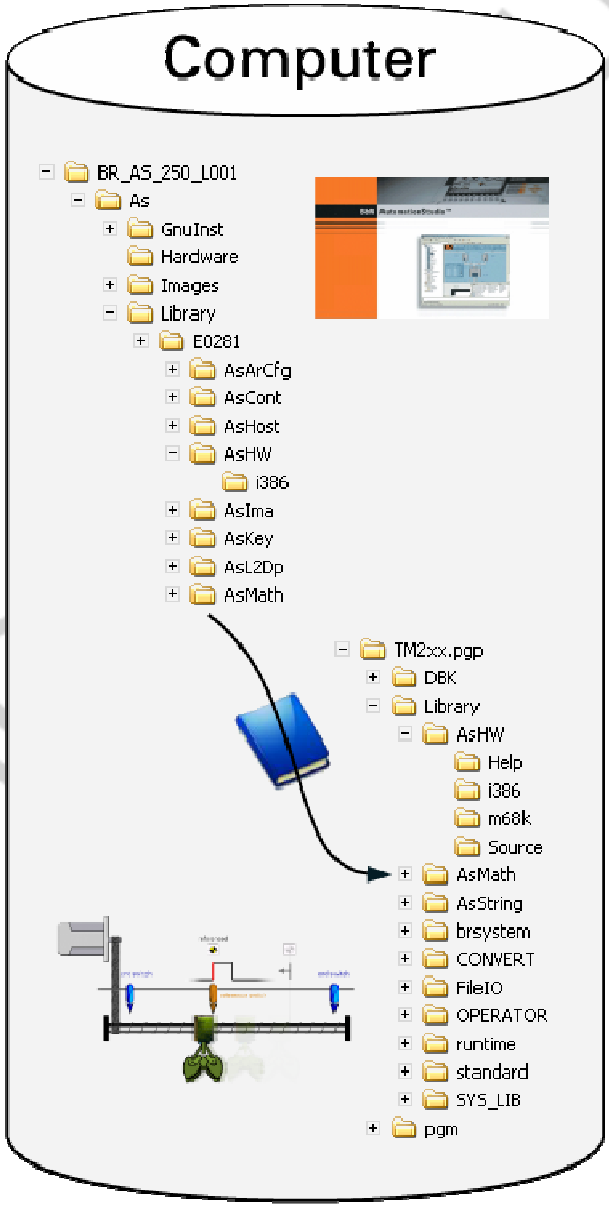


Fig. 9 Storing libraries on the PC

2.2.5 Library dependencies

The range of functions and the way that libraries work can be tied to specific **conditions**.

For example, a library can be associated with a specific **Automation Runtime version**. In this case, the functionality of the library is based on Automation Runtime. If a change is made to the Automation Runtime, then the corresponding libraries in the project are also changed automatically. To guarantee upwards compatibility, functions, function blocks, data types and constants are included in all following versions. Only expansions can be made to the range of functions.

Note:

Libraries that are not dependent on Automation Runtime are not automatically exchanged when changing versions.

Dependencies from other libraries are also possible. These are mostly present when creating **user libraries**, if functions or function blocks from another library are used (see "User libraries"). However, **standard libraries** can also have dependencies on one another.

Caution:

When adding standard libraries, make sure that they **only come from the Automation Runtime version set in the project**.

2.3 Help

The Automation Studio online help contains detailed descriptions of the topics "Libraries" and "Library Manager" which can be found under **Automation Software:Automation Studio**. The online help can be accessed by either using the Help icon in the toolbar or by pressing the F1 key.
The Help system is context-sensitive (i.e. the description of the currently selected element is automatically displayed when the help system is opened with the Help icon or the F1 key).

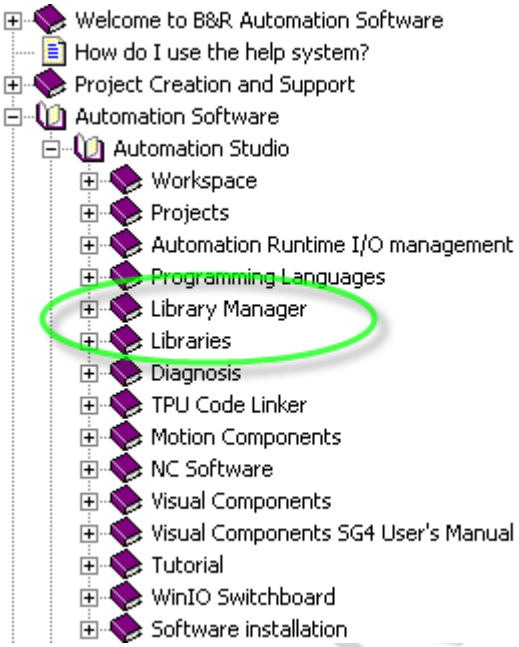


Fig. 10 Help for the Library Manager and Libraries

Exercise: TON, CTU, AsString

Open the Automation Studio online help and search for information about the "TON" and "CTU" function blocks.

Select the "strcmp" function from the "AsString" library and press the F1 key or the Help icon.

2.4 Using functions and function blocks

In principle, functions and function blocks are used the same way **on all B&R target systems** and behave the same way.

However, there are also functions and function blocks that are used exclusively on one of the two platforms because they are tied to a specific system architecture.
For example, file management is not supported for SG3 controllers. As a result, the function blocks for file management from the "FileIO" library are not available there either.

2.4.1 Adding functions or function blocks

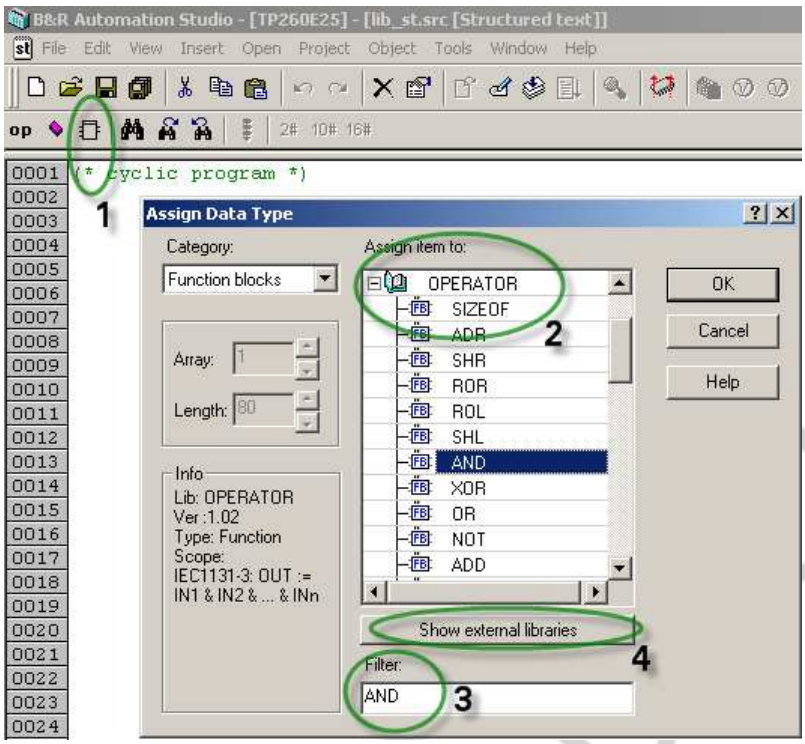


Fig. 11 Adding a function or a function block in IEC languages

1 Select the "Insert function" button to add a function a function block in an IEC programming language or Automation Basic.

2 This opens a dialog box that displays the libraries integrated in the project. The functions and function blocks contained in the libraries can be selected from here.

3 If you know the name of the function or function block you're looking for, you can enter it in the filter to locate it quickly.

4 All of the standard B&R libraries are displayed by selecting the button "Show external libraries". If a function or a function block from an external library is selected, then the library is automatically copied to the project.

2.4.2 Syntax for functions in the "pow()" example

This function from the "AsMath" library is used to add exponents to a value (e.g. x^y).

Ladder Diagram

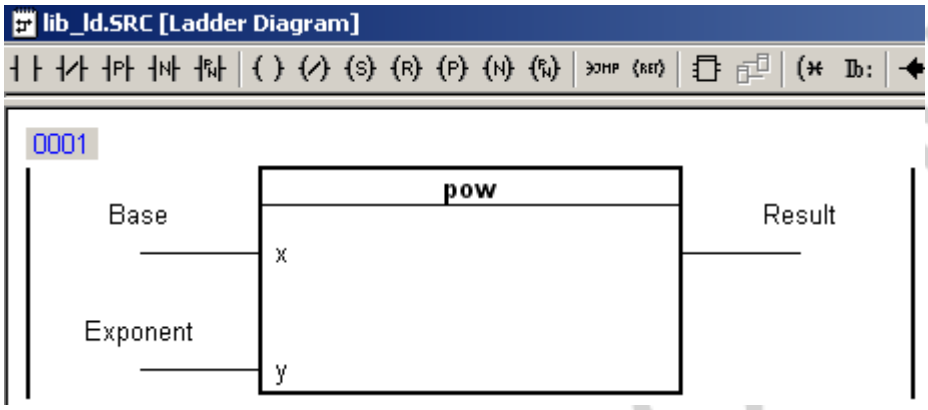


Fig. 12 Function in the Ladder Diagram

Structured Text

The function call appears as follows when inserting the function using the dialog box. The inputs (x and y) are only indicated symbolically and must be replaced by corresponding variables.

```
(* Function name(input1, input2) *)
pow(x, y);
```

The function can also be executed directly without the dialog, like an operator.

```
(* Result := function name(Basis, Exponent) *)
Result := pow(Base, Exponent);
```

Automation Basic

The function call appears as follows when inserting the function using the dialog box. The inputs (x and y) are only indicated symbolically and must be replaced by corresponding variables.

```
; Function name(input1, input2)
pow(x, y)
```

The function can also be executed directly without the dialog, as an operator.

```
; Result = function name(Basis, Exponent)
Result = pow(Base, Exponent)
```

ANSI C

In ANSI C, there is no dialog box for inserting functions or function blocks. The call can be copied here from the corresponding header file (in this case e.g., "asmath.h" or "math.h").

```
/* Output data type function name(data type input1, data
type input2) */
float pow(float x, float y);
```

```
The function call appears as follows when using the
variables from the example:
```

```
/* Result = function name(Basis, Exponent) */
Result = pow(Base, Exponent);
```

2.4.3 Syntax for function blocks in the "CTU()" example

The "CTU()" function block from the "standard" library is an upward counter.

Ladder Diagram

Unlike a function call, a name for the function block instance must be assigned here ("Counter1").

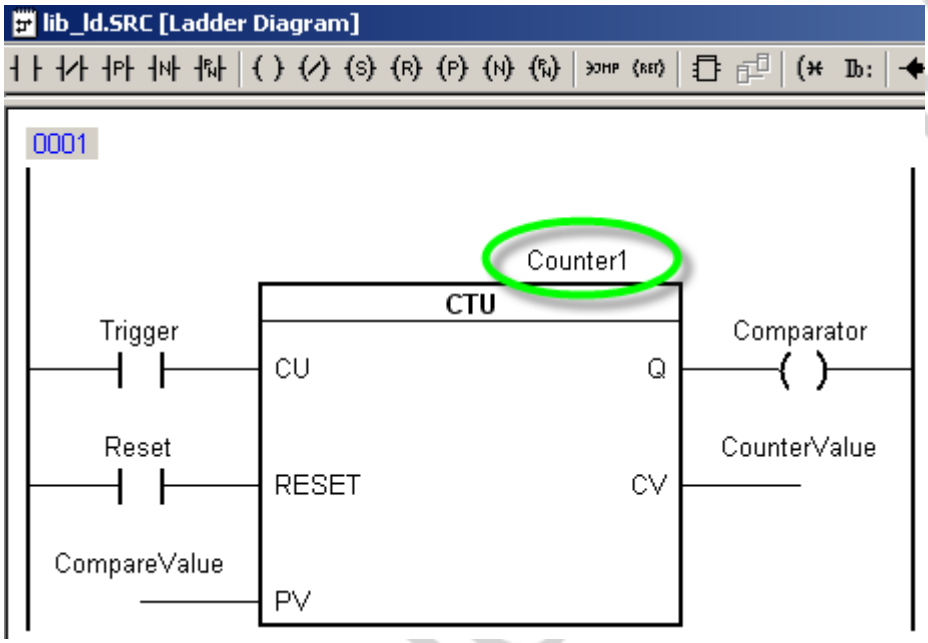


Fig. 13 Function block in the Ladder Diagram

Structured Text

The function call appears as follows after assigning the instance name when inserting the function block using the dialog box. The inputs (CU, RESET and PV) are components of the instance and must be written with the corresponding variables or values when setting the parameters.

The instance of the function block "Counter1" is a structure variable with the data type "CTU".

```
(* Function block name(input1:=,
                       input2:=, input3:=) *)

Counter1(CU:=, RESET:=, PV:=);
```

A complete function block call could appear as follows:

```
(* Function block parameter settings and call*)
Counter1(CU:= Trigger, RESET:= Reset,
         PV:= CompareValue);

Evaluation of the outputs *)
Comparator:= Counter1.Q;
CounterValue:= Counter1.CV;
```

The function block parameter setting can also be handled as follows in order to maintain a clear and organized structure when dealing with multiple input parameters or long variable names:

```
(* Parameter settings *)
Counter1.CU:= Trigger;
Counter1.RESET:= Reset;
Counter1.PV:= CompareValue;

(* Function block call *)
Counter1();

(* Evaluation of the outputs *)
Comparator:= Counter1.Q;
CounterValue:= Counter1.CV;
```

Automation Basic

The function call appears as follows after assigning the instance name when inserting the function block using the dialog box. The inputs (CU, RESET and PV) are components of the instance (structure variable) and must be written with the corresponding variables or values when setting the parameters.

The instance of the function block "Counter1" is a structure variable with the data type "CTU".

```
Counter1.CU=        ; Parameter setting
Counter1.RESET=
Counter1.PV=
Counter1 FUB CTU() ; Function block call
```

A complete function block call could appear as follows:

```
Counter1.CU = Trigger; Parameter setting
Counter1.RESET = Reset
Counter1 FUB CTU()      ; Function block call

; Evaluation of the outputs
Comparator = Counter1.Q;
CounterValue = Counter1.CV;
```

In Automation Basic, the function block can also be called in a line. This method eliminates the instance, which is why all input and output parameters must be inside the brackets.

```
; Parameter setting, call and evaluation
; of the outputs in a line
CTU(Trigger, Reset, CompareValue, Comparator, CounterValue)
```

In order to maintain the **clear and organized structure**, **you should generally avoid using this method** because there is no clear separation between inputs and outputs. This **could make debugging more difficult** when dealing with more complex function blocks.

ANSI C

In ANSI C, there is no dialog box for inserting functions or function blocks. The information necessary for the call can be copied here from the corresponding header file (e.g. "standard.h").

```
/* Variable declaration */
BOOL      Trigger, Reset, Comparator;
UINT      CompareValue, CounterValue;

/* Function block instance */
CTU_typ   Counter1;

/* Parameter setting */
Counter1.CU = Trigger;
Counter1.RESET = Reset;
Counter1.PV = CompareValue;

/* Function block call */
CTU(&Counter1);

/* Evaluation of the outputs */
Comparator = Counter1.Q;
CounterValue = Counter1.CV;
```

3. STANDARD LIBRARIES

B&R provides a **comprehensive package of standard libraries** with Automation Runtime.

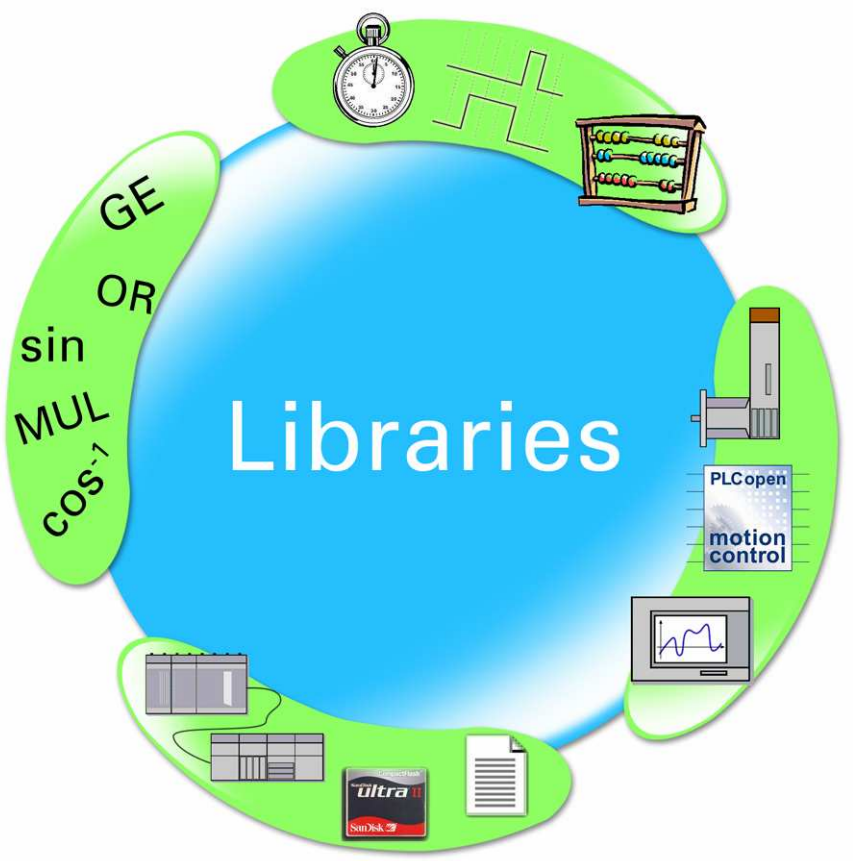


Fig. 14 Function range of standard libraries

The function range of these standard libraries begins with simple functions and function blocks, which are not contained in the standard code of the respective programming language or which can be deleted with simple loops.
Examples of this include the following:

- Timer (delays), counter, edge detection
- String processing
- Arithmetic
- Logic operations

Highly complex and powerful functions and function blocks are also contained, which considerably minimize the development effort required for applications and save a great deal of program code.
Here a few examples of this:

- Control algorithms
- Data objects and file management
- Webserver data exchange
- Network functions
- Axis control
- Graphics functions

Note:

Standard libraries are always **binary libraries** (i.e. there is no source code for the functions and function blocks from standard libraries).

Using functions and function blocks from standard libraries provides the following advantages:

- As complete program units, they are **fully tested** and functionality is checked.
- They are **serviced by B&R** and the range of functions is continually updated and expanded.
- **Uniform documentation with examples** in different programming languages is integrated in the Automation Studio online help.
- Use of the functions and function blocks is uniform and user-friendly thanks to compliance with both international as well as B&R-internal **standards and certifications**.

Note:

A complete overview of all of the B&R standard libraries can be found in the appendix.

3.1 Function blocks with enable input and status output

Function blocks that handle more complex tasks or that require multiple cycles for processing, have an enable input and a status output.

The **enable** input can be used to switch the function block **on or off** (0 → off, 1→ on). This can reduce the load on the CPU as long as the function block does not have to be processed constantly.

The **status** output provides information about the **processing status** of the function block.

Status number	Meaning
0	Function block executed without error.
65535	Function block processing not yet complete (busy). Call again in the next cycle.
65534	Enable input not set. The function block is switched off.
Other number	An error occurred while processing the function block. Explanations of the error numbers can be found in the Automation Studio online help for the respective library.

Example

The "HwGetBatteryInfo" function block from the "AsHW" library is used to check the charge of the backup battery.

The "BatteryInfo" instance is a "HwGetBatteryInfo" type.

```
USINT     Battery status;
BOOL      ReadBattStatus;
STRING    Device[16];

(* Parameter setting *)
Device:= ´SL0.SS0.HW´;

BatteryInfo.enable:= ReadBattStatus;
BatteryInfo.pDevice:= ADR(Device);
BatteryInfo.ordinal:= 2;

(* Function block call *)
BatteryInfo();

(* Evaluation of the status *)
IF BatteryInfo.status = 0 THEN
    (* Evaluation of the output *)
    BatteryStatus:= BatteryInfo.state;
    ReadBattStatus:= FALSE;
ELSIF BatteryInfo.status <> 0 AND
      BatteryInfo.status <> 65535 THEN
    ReadBattStatus:= FALSE;
    (* Error correction *)
    ...;
END_IF
```

3.2 Address inputs

Some functions and function blocks require the address of the memory, where the input parameters are located, as input. As in the above example, this can be the address of a variable or the address of a freely allocated memory area.

These methods for parameter transfer are used if the length of the data (number of bytes) is not already known, but should not be limited.

In the previous example, the device name for different CPUs was able to have different lengths. This is irrelevant for the function block because only the address of the string is always transferred.

3.3 Limitations on the Init subprogram or cycle program

Some functions or function blocks require a high amount of system resources to execute highly complex tasks such as the initialization of hardware.

Therefore, these functions and function blocks should only be called in the Init subprogram to prevent causing cycle time violations.

Note:

Information about these types of limits can be found in the description in the Automation Studio online help for the respective library, function or the respective function block.

There are also function blocks that are processed asynchronous to the cyclic task class system. These are mostly function blocks that access hardware and must wait for a response from the operating system (e.g. access to the Compact Flash). Therefore, they must be executed over multiple cycles, until they output a value to the status output that is unequal to 65535 (Busy).

As a result, these types of function blocks are mostly processed in cyclic mode.

4. USER LIBRARIES

User libraries can be created in Automation Studio using the Library Manager. User libraries can contain functions and function libraries that are specially programmed by the user **according to the application requirements**.

> **Note:**
>
> User libraries are also called **source libraries** because the source code is usually kept in the project. Depending on the type of library, this can also be an **IEC or ANSI C library**.
> A binary library can also be made from a source library using basic methods (described later in this section).

The advantages of using libraries that were mentioned earlier also apply to user-specific libraries.
User libraries also have the following additional advantages:

- Just like those in standard libraries, functions and function blocks from user libraries are also used to structure and limit the amount of necessary program code.
- This makes it possible to **isolate** program sections that are complex or are used frequently from the actual program and package them into user-specific functions and function blocks.
- This can considerably improve the overall organization and clarity and makes it **much easier** to **pass on or reuse** the functionalities.

4.1 Creating user-specific libraries

The creation of user-specific libraries involves a few steps that are always performed based on a fixed procedure.

- First you must decide whether the functions and function blocks should be implemented in **IEC** programming languages (including Automation Basic) **or in ANSI C**.
- Functions and/or function blocks are then added and the **source code** is created.
- **Tests and error corrections** are essential for flawless execution of the functions and function blocks.
- A self-made online help **documentation** can be created for the library.
- Further steps must taken in order to **pass the library on** as either a source or binary library.

All of these points will be addressed in great detail later in this section.

Creating a library

The same wizard that was used to integrate existing libraries is also used to create a new library. You must first begin by pressing the "Insert Library" button.

The library must be given a **name with a maximum of 8 characters**.

The **type** determines the programming languages that will be possible for the implementation of the functions and function blocks. If "C-Library" is selected, then functions and function blocks must be programmed in ANSI-C. However, if an "IEC-Library" is created, then the programming languages Instruction List, Structured Text and Automation Basic are available.

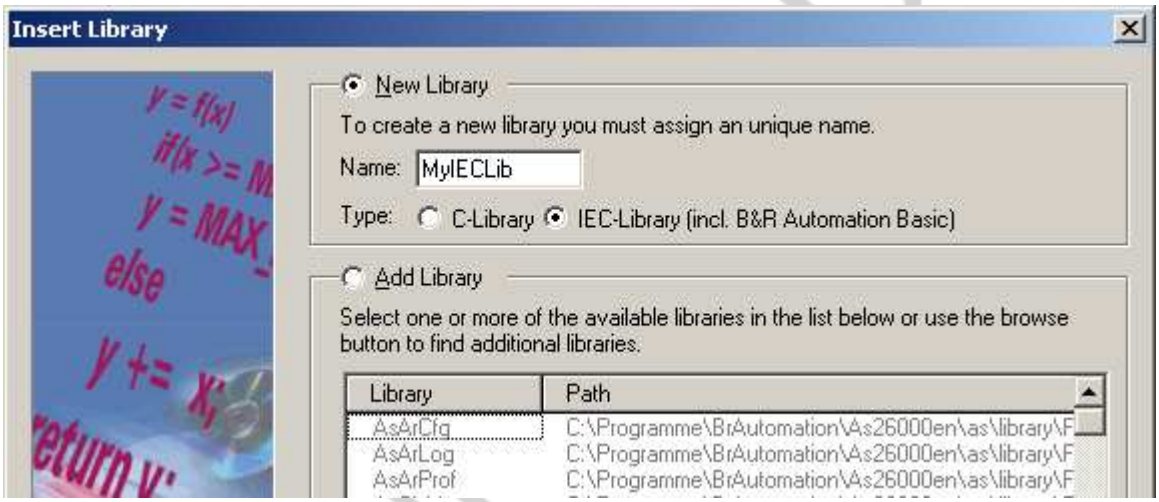


Fig. 15 Creating a new library

4.2 IEC library

4.2.1 Creating a function

A function or a function block can be create using the "Insert Function / Function Block") button or by right-clicking on the library.

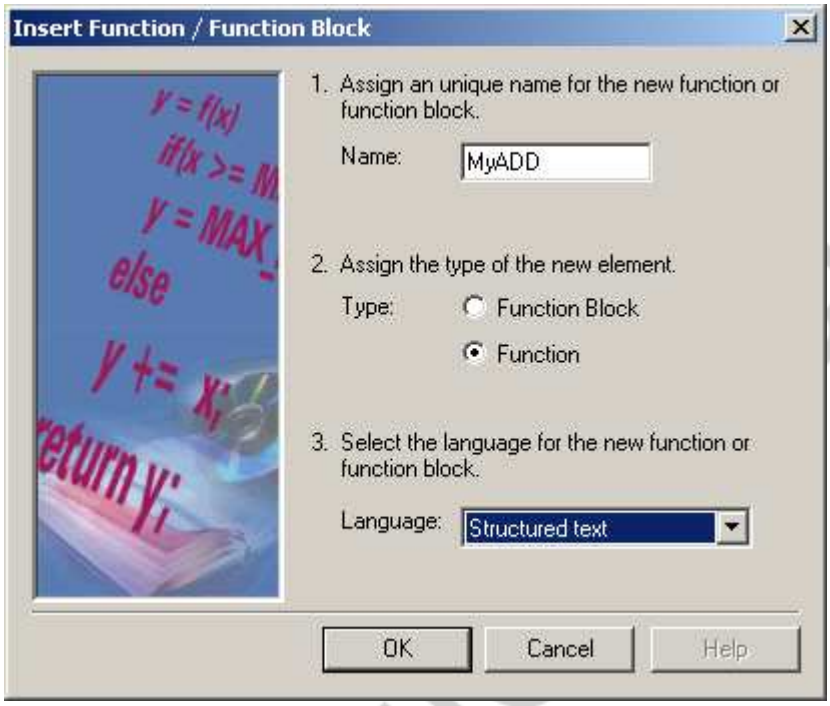


Fig. 16 Inserting a function or function block

A **unique name** that has not yet been used in any other module must be given for the function.

"Function Block" or "Function" can be selected as type.

The **programming language** for the respective function can be selected under "Language".

After confirming the selection with "OK", the function is created and you can go on to **declare the interface** and **create the source code**.

At this point, the function should e.g. add the two inputs and write the result to the output.

Additional inputs can be inserted using the "Insert Declaration" button or by right-clicking in the declaration.

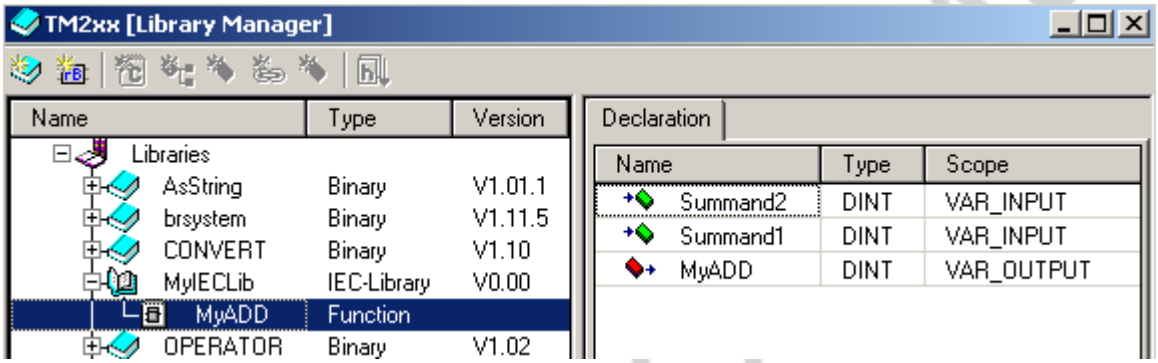


Fig. 17 A function in the Library Manager

The source code editor is opened by double-clicking on the function icon.

The source code for this function looks like this:

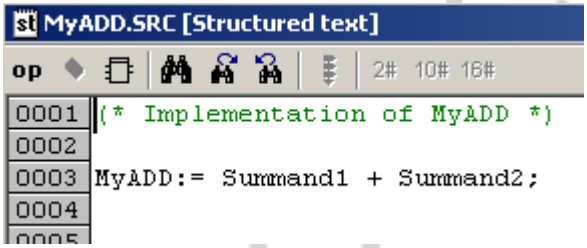


Fig. 18 Source code for the "MyADD" function

Task: Build your own function block

Program a function, which limits an input value to a minimum or maximum.

The output value is calculated as follows:

If the minimum value is greater than the maximum value, then the output value is equal to the maximum value.

If the input value is less than the minimum value, then the output value is equal to the minimum value.

If the input value is greater than the maximum value, then the output value is equal to the maximum value.

If the input value is greater than the minimum value and less than the maximum value, then the output value is equal to the input value.

Name: MinMax

Inputs:
```
DINT      Lower, In, Upper
```

Output:
```
DINT      [Function name]
```

4.2.2 Interface for the functions and function blocks

The interface for a function or function block is comparable with the **variable declaration** in cyclic tasks. It contains all of the necessary input, output and internally used variables. Unlike cyclic tasks, there are additional options when declaring variables here.

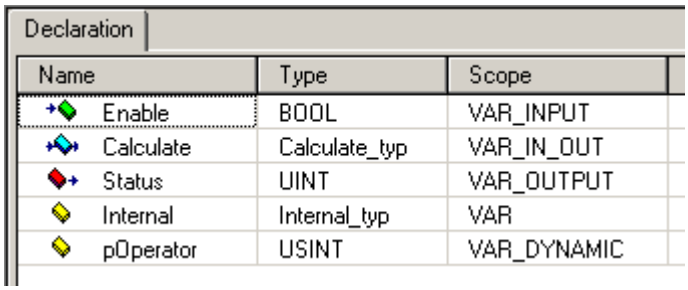


Fig. 19 Interface declaration

VAR_INPUT:
Input parameter.

VAR_OUTPUT:
Output parameter.

VAR:
Static internal variable.

VAR_DYNAMIC:
Dynamic internal variable without an external reference. This variable can only be referenced to variables in a function or function block's declaration.

VAR_IN_OUT:
The parameters are applied to the executed function block as pointers to their memory location (i.e. the specified variable is passed onto the function block so that it can be read and changed there). As a result, changes are automatically applied to the variable declared outside of the executed function block.

VAR_INPUT_DYNAMIC:
Dynamic input/output parameter. Dynamic inputs get the appropriate pointer by means of ADR function. The variable declared outside of the function block is used directly to edit the function block.

4.2.3 Creating a function block

Essentially the same process used to create a function is also used to create a function block.
The only difference is that "Function Block" must be selected as type.

Task:

Program an upwards counter.

Function description:

When the "Reset" input is TRUE, then "CounterValue" is set to 0.

Otherwise, the "CounterValue" is incremented by 1 with each increasing edge on the "Trigger" input.

The "Comparator" output is TRUE if the "CounterValue" output is greater than or equal to the "ComparatorValue".

Inputs:
```
BOOL Trigger, Reset
DINT ComparatorValue
```

Outputs:
```
BOOL Comparator
DINT CounterValue
```

Internal:
```
BOOL EdgeMarker
```

Note:

The internal variable "EdgeMarker" is required as an auxiliary variable for detecting an increasing edge on the "Trigger" input. Alternatively, the "R_TRIG()" function block from the "standard" library can also be used. It must first however, be declared an internal variable.

4.2.4 Library properties

When working with standard libraries or binary user libraries, this dialog only displays the properties. When working with IEC or ANSI-C libraries, this dialog can be used to set the properties.

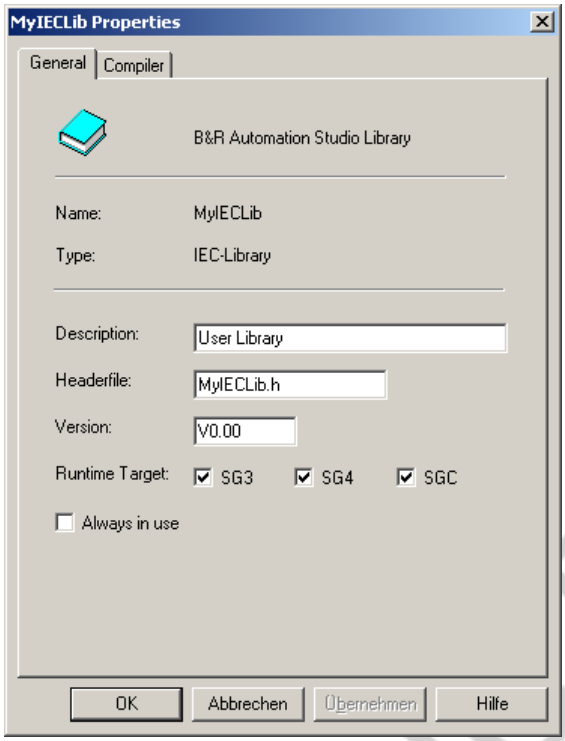


Fig. 20 Properties for an IEC user library

Description:
Brief description of the library's functionalities.

Header file:
Name of the header file (*.h file) created for the library.
The name of the library is entered automatically.

Version:
Version identification of the library in the form "Vx.yy.z".

Runtime target:
Determines which target platform(s) the library was/should be created (compiled) for.

4.2.5 Additional Dependencies

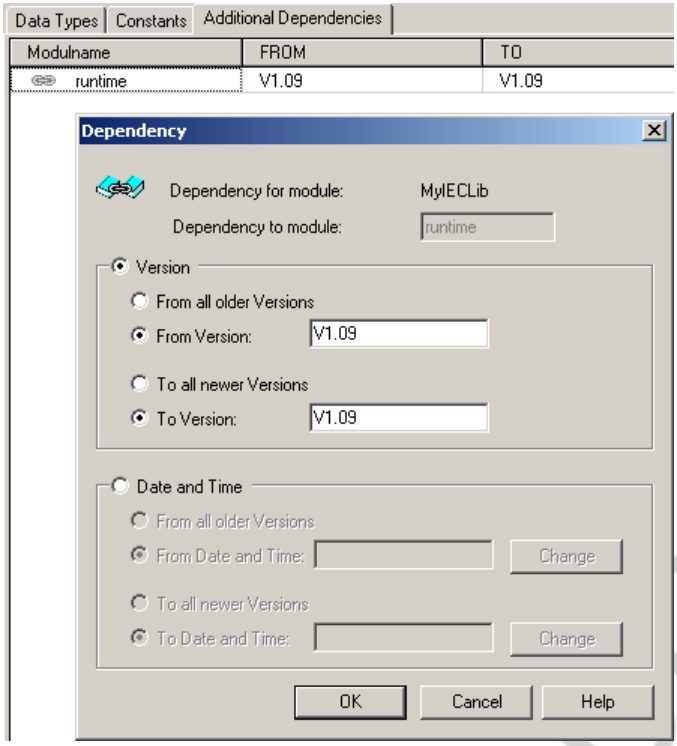


Fig. 21 Additional dependencies of a user library

If functions or function blocks from another library are used, then an **additional dependency** to that library is automatically entered. That means if this library is used in a project, then all other libraries containing a dependency must also be integrated in the project.

Different options are available for setting the dependency. It can be limited by date or specific version (or a minimum and maximum version number).

4.2.6 Debugging and error diagnostics options

Mistakes can be made anytime and anywhere that humans are at work. As a result, certain parts of a program sometimes may not react exactly as they're expected to when programming.

A debugger has been integrated into Automation Studio to make it easier to locate the cause of unforeseen program errors. This tool allows users to easily locate errors in all software sections of an application.

An online connection to the controller where the software section that needs to be tested is located is required to run the debugger.

Monitor mode  must be started in order to use the debugger.
A new group is now displayed in the toolbar, which contains functions for the debugger. 

A breakpoint in the program can be set using the  button or by clicking on the line number. When this line is reached during processing, the program is stopped before being executed and the current variable values can be checked.

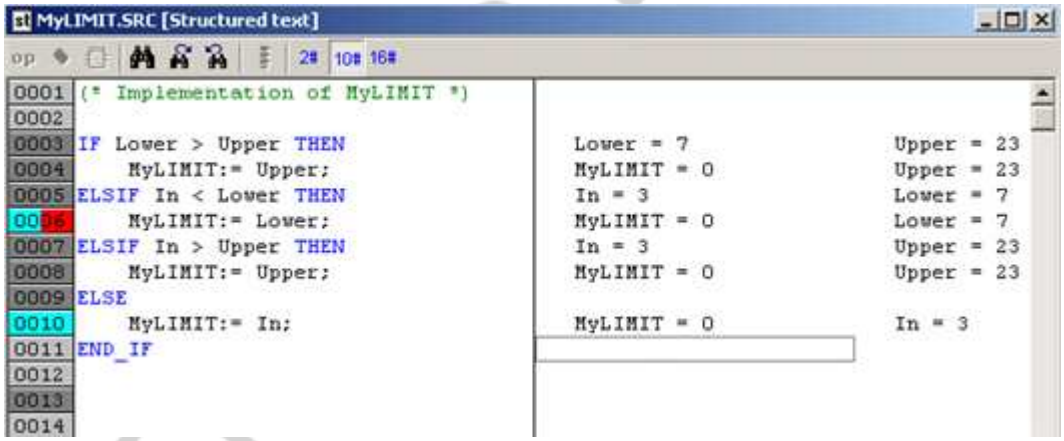


Fig. 22 A function in debug mode with breakpoint

Note:

Breakpoints can only be set on lines with numbers that are shown on a dark gray background.

The program is continued to the next breakpoint by clicking on the "Continue"  button. The "Step Into"  button and "Step Over"  buttons are used to continue the program step-by-step.

Note:

A more detailed description of the debugger would go well beyond the framework of this training manual. Such a description can be found in the Automation Studio online help under **Automation Software:Automation Studio:Diagnosis:Debugger**.

4.2.7 Testing with AR000

AR000 is an Automation Runtime system based on Windows-32 which is not real-time capable, but essentially corresponds to the functionality of all controllers. Since the **AR000 is conceived for testing purposes**, no hardware is used.

Therefore, functions and function blocks that do not directly access the hardware or are not designed for specific hardware configurations can be easily tested on the AR000.

The AR000 is started using the menu **Tools:AR000**. Once the online connection has been set on the AR000, it can be used like a real SG4 controller and can be used to test the application.

Note:

For the most part, functions and function blocks can be sufficiently tested using the AR000 because their functionality is not generally limited to specific hardware configurations.

4.3 ANSI C libraries

There are a few differences in the way that ANSI C and IEC user libraries are used in the Library Manager.

For example, when using ANSI C libraries, the source code files can be managed right in the Library Manager. The "Source" tab on the right side of the Library Manager is used to do this.

4.3.1 Creating an ANSI C library

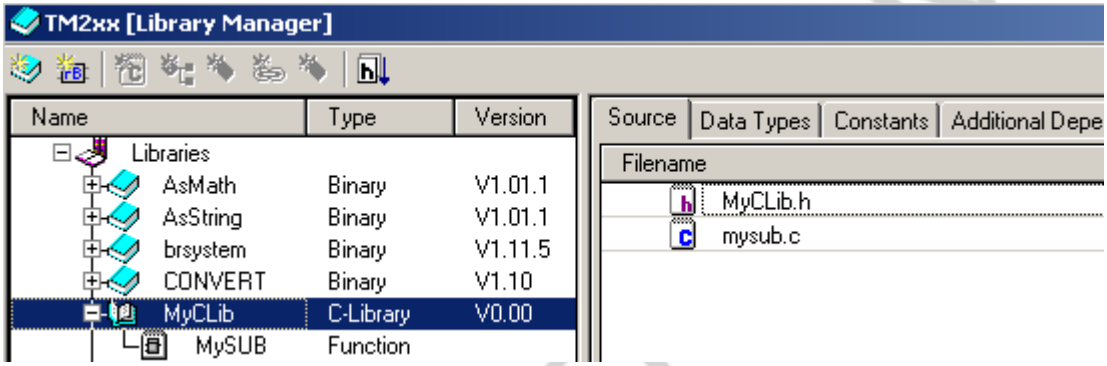


Fig. 23 ANSI C library in the Library Manager

The corresponding header file "MyCLib.h" is automatically created when creating an ANSI C library. All declarations for functions, function blocks, constants and data types are entered here. This file is write-protected and automatically updated when compiling or by manually generating via the "Generate *.a, *.h file" button .

4.3.2 Functions and function blocks

In ANSI C libraries, the creation of functions and function blocks and the declaration of their interfaces is done the same way as in IEC libraries.

mysub.c [C Language]

```
#include <MyCLib.h>

/* Implementation of MySUB */

signed long MySUB(signed long Minuend, signed long Subtrahend)
{
    return(Minuend - Subtrahend);
}
```

Fig. 24 Source code for the MySUB function

The files for the source code must be created manually using "Insert:File". Before the implementation of the source code for the function of function block, the header file must be included for the library.

MyCLib.h [C Language]

```
/* Datatypes of function blocks */


/* Prototyping of functions and function blocks */
signed long MySUB(signed long Minuend, signed long Subtrahend);
```

Fig. 25 Prototype of the function in the library's header file

The prototype of the function of function block (call) can be copied from the header file. The implementation is then inserted between curved brackets in the source file.

Note:

When using ANSI C libraries, it is quite easy to break up the work of developing the functionalities. The source files for the functions and function blocks can be easily copied out of the library directory, edited somewhere else, then copied back.

4.3.3 Characteristics

In comparison to the IEC libraries, ANSI C libraries have a few additional properties.

Source file properties

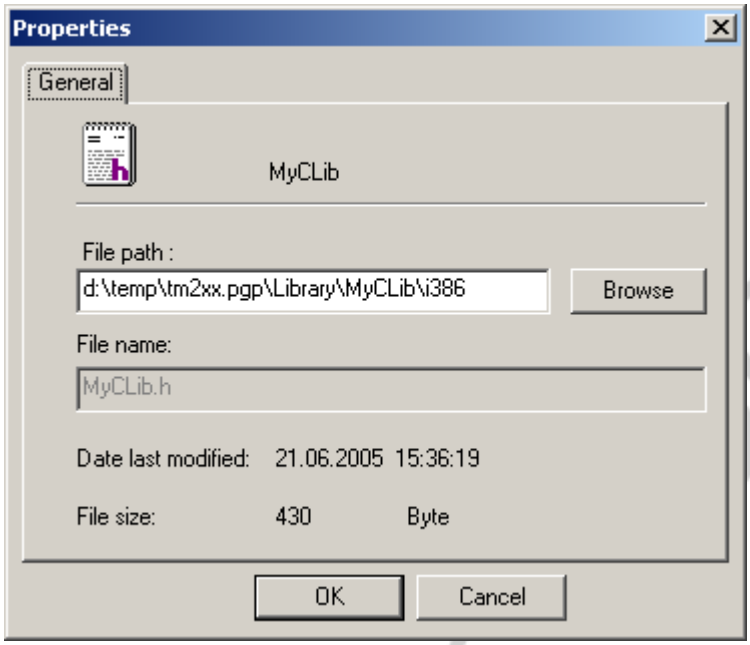


Fig. 26 Source file properties

The file path can be entered here. When using ANSI C libraries, the source files can be stored in any directory. By default, the directory of the library in the project where it was created is entered.

Additional ANSI C library properties

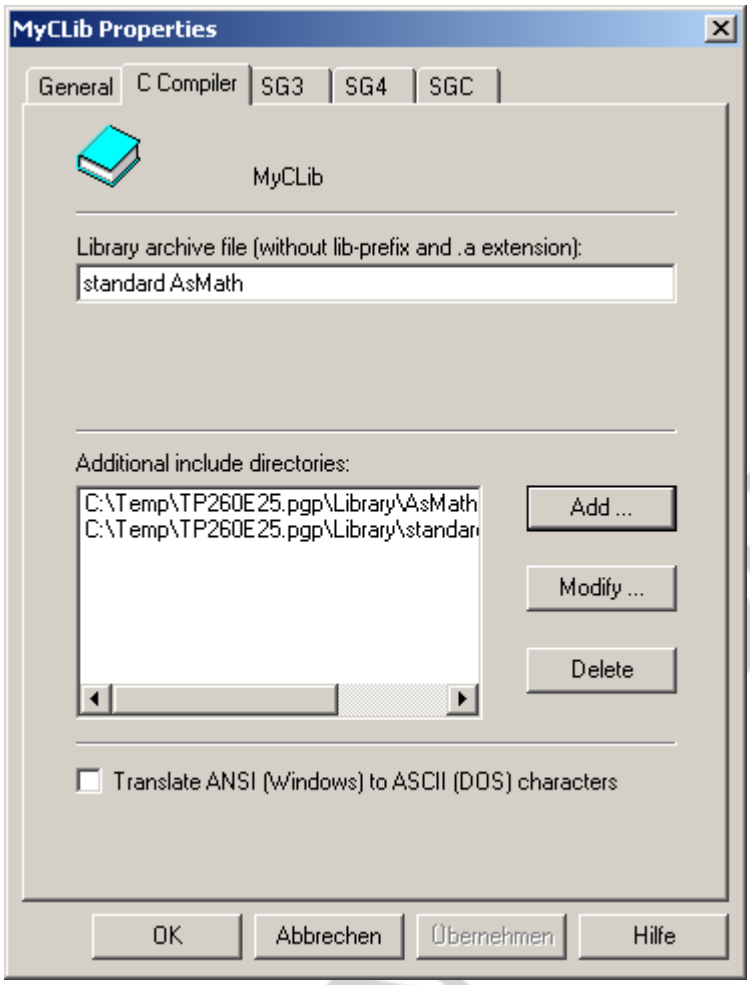


Fig. 27 Library properties

If functions and function blocks from other libraries are use, the names and paths of the archive files (*.a files) must be entered in this tab. For example, the libraries "standard" and "AsMath" are used here.

Note:

Detailed information about creating and using ANSI C libraries can be found in the Automation Studio online help under **Automation Software:Automation Studio:Library Manager:Creating a New Library**.

4.4 Creating a user library help system

A separate online help system can be created for each user library, and opened using the F1 key or the help icon  .

The following guidelines must be observed:

- The help system for must be available as .chm files (compiled HTML help files). These can be created using diverse programs available on the market (Microsoft® HTML Help WorkShop, RoboHelp®, FAR).
- The chm. files for the user libraries must contain the prefix "Lib" (e.g. LibMyIECLib.chm) and be stored in the library sub-directory [LibraryName]\Help (e.g. "MyIECLib\Help").
- when creating the .chm file the following directory structure must be maintained as shown here in the example MyIECLib:

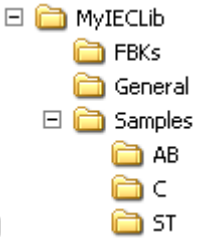


Fig. 28: Directory structure for creating the .chm file

- Therefore, the root directory for creating the .chm file has the same name as the library. This directory must contain the subdirectory "FBKs", where the files are kept for describing the functions and function blocks. The descriptions must each be created in a separate .html file (not .htm files!) with the name of the function or function block.

The other subdirectories are only examples and are optional.

4.4.1 Passing on user libraries and creating binary libraries

The library is created in the project with the name of the library in the form of a directory (in this case e.g. "MyIECLib"). This library contains all of the necessary files and objects that belong to the library.
Only this directory has to be copied from the project in order to pass on a library.

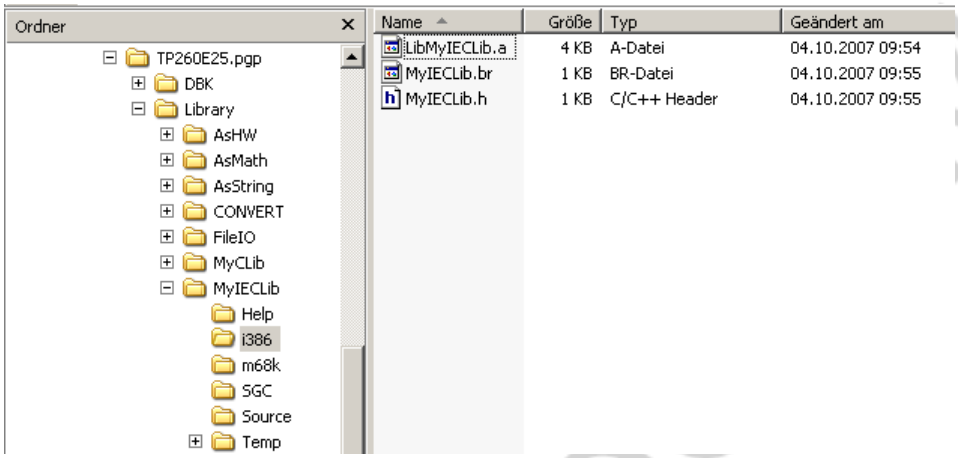


Fig. Directory structure of the user library in the project directory

The user library directory is divided into five subdirectories:

Help:
Online help, if available.

i386:
Compiled files for SG4 (Intel platform) (*.a, *.h, *.br).

m68k:
Compiled files for SG3 (Motorola platform) (*.a, *.h, *.br).

SGC:
Compiled files for SGC (Compact CPUs) (*.a, *.h, *.br).

Source:
The files in this directory contain the source code for the functions and function blocks.

Temp:
Temporary files.

The following steps are necessary to make a binary library from an IEC or ANSI-C library:

- Copy the library to another directory.
- Delete all files from the library directory (e.g. "\MyIECLib\...").
- Delete the entire "\Temp" directory.
- Delete all files from the "\Source\..." directory.
- Insert the library again using the Library Manager

Note:

When using binary libraries, the number of functions or function blocks and how they are implemented cannot be changed. That's why it is always a good idea to make backup copies of your user libraries.

5. SUMMARY

Libraries are an extremely useful and powerful to when creating software. Using functions and function blocks from standard libraries can considerably simplify and reduce the amount of work required to develop an application. The improved structure makes it much easier to read the shorter program code created. Furthermore, libraries are encapsulated program sections that are much easier to transfer to other projects and can be reused.
User libraries allow the creation and organization of individual user-specific functions and function blocks.

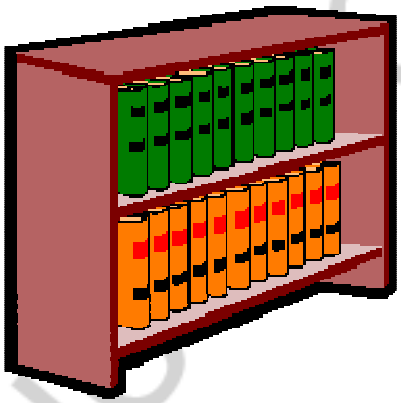


Fig. 29 Library

Participants are familiar with the use of libraries in Automation Studio and are aware of their benefits.
By skillfully navigating the help system, participants can quickly locate and effectively use the documentation and descriptions of the necessary functions or function blocks.
Participants can also create their own library and functions / function blocks according to their individual demands.

With their knowledge of how to work with libraries, functions and function blocks, participants are also able to structure complex applications in a clear and organized manner, thereby increasing readability and effectiveness.

6. APPENDIX

6.1 Overview of the B&R standard libraries

Library	Short description
Acp10_mc	Motion function block for ACOPOS drives, specified in PLCopen - Technical Committee 2 –Task Force "Function blocks for motion control V1.0"
AsArCfg	Reading and writing Automation Runtime configuration settings
AsArProf	Operation of the AR profiler from an application
AsCont	Support of hardware modules
AsHost	Conversion of IP addresses to domain names and vice-versa
AsHW	Reading information from the respective target system
AsIMA	Activation of the INA2000 manager
AsIO	Determining the states and values of a data point, force handling
AsIOAcc	Reading and writing access to non-cyclic I/O data points
AsIODiag	Creation and evaluation of IO module information
AsIOMMan	Activation and deactivation of IO configuration modules or mapping modules
AsKey	Functions for querying the dongle
AsL2DP	Operation of the L2DP slave module 3IF661.9 and 3IF761.9
AsMath	Mathematic functions not covered by the "OPERATOR" library
AsMem	Managing memory blocks in large memory areas (memory partitions)
AsPlkSup	Access to different configuration words for the 2003 screw-in modules
AsString	Memory manipulation and string handling
AsTime	Support of date and time functions on the controller
AsUPS	Communication with a UPS
AsWeigh	Function blocks for using strain gauge modules (e.g. AI261).
AsWStr	Processing of 16-bit wide character (Unicode) strings
BRSystem	Operation of the CPU
C220man	Operation of the panel controllers
CAN_Lib	Operation of the CAN controller
CANIO	Operation of B&R2003 CAN nodes
Commserv	System expansion for INAcInt library
CONVERT	Conversion functions according to IEC61131-3
DataObj	Handling of data objects
DM_Lib	Storage of data objects in nonvolatile memory
DRV_3964	3964R protocol
DRV_mbus	Modbus protocol

DRV_mn	MiniNet protocol
DVFrame	Frame driver library for serial interface operation
Ethernet	Data exchange via UDP or TCP
EthSock	Integration of socket functions
FDD_Lib	Operation of the external floppy disk station MFDD70S
FileIO	File and directory management
IF361	Operation of the IF361 interface module (Profibus DP Slave)
IF661	Operation of IF661 interface module (Profibus DP Slave)
INAclnt	INA2000 client communication
IOConfig	Execution of shovel tasks on 2003
IOCtrl	Operation of 2003 IOs
IO_lib	Operation of the I/O modules
Logging	Operation of the profiler from an application
LoopConR	Implementation of tasks for control technology (calculation with REAL values)
LoopCont	Implementation of tasks for control technology (calculation with Integer values)
NET2000	NET2000 protocol
OPERATOR	Operators according to IEC61131-3
PB_lib	Profibus protocol (FMS)
PowerLnk	Handling of the Powerlink interface board IF686.
PPdpr	Exchange of data between CPU and PP
RIO_lib	Operation of remote I/O
runtime	Internal support functions and function blocks
Spooler	Allows the spooling of data on IPs
SRAM200x	Functions for handling the SRAM200x
Default	IEC61131-3 standard functions and function blocks
SYS_lib	Various system functions
TCPIPMGR	Data exchange via UDP or TCP
VCScrsht	Saving a current image of the target system as a bitmap (*.bmp)
visapi	Programming interface for controlling the visualization while the system is running
VNCServ	Visualizations that run on SG4 targets and support VGA, can be viewed on the PC

6.2 Solution to task 4.1.2

Program a function, which limits an input value to a minimum or maximum.

Solution in Structured Text:

```
IF Lower > Upper THEN
     MyLIMIT:= Upper;
ELSIF In < Lower THEN
     MyLIMIT:= Lower;
ELSIF In > Upper THEN
     MyLIMIT:= Upper;
ELSE
     MyLIMIT:= In;
END_IF
```

6.3 Solution to task 4.2.3

Program an upwards counter.

Solution in Structured Text:

```
IF Reset = TRUE THEN
     CounterValue:= 0;
ELSIF Trigger = TRUE AND EdgeMarker = FALSE THEN
     CounterValue:= CounterValue + 1;
     EdgeMarker:= TRUE;
ELSIF Trigger = FALSE THEN
     EdgeMarker:= FALSE;
END_IF

IF CounterValue >= ComparatorValue THEN
     Comparator:= TRUE;
ELSE
     Comparator:= FALSE;
END_IF
```

Overview of training modules

TM200 – B&R Company Presentation **
TM201 – B&R Product Spectrum **
TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM212 – Automation Target **
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LAD)
TM241 – Function Block Diagram (FBD)
TM246 – Structured Text (ST)
TM247 – Automation Basic (AB)
TM248 – ANSI C
TM250 – Memory Management and Data Storage
TM260 – Automation Studio Libraries I
TM261 – Closed Loop Control with LOOPCONR

TM400 – The Basics of Motion Control
TM410 – The Basics of ASiM
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM445 – ACOPOS ACP10 Software
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors

TM500 – The Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER

TM600 – The Basics of Visualization
TM610 – The Basics of ASiV
TM630 – Visualization Programming Guide
TM640 – ASiV Alarm System
TM650 – ASiV Internationalization
TM660 – ASiV Remote
TM670 – ASiV Advanced

TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM730 – PVI OPC

TM800 – APROL System Concept
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming
TM890 – The Basics of LINUX

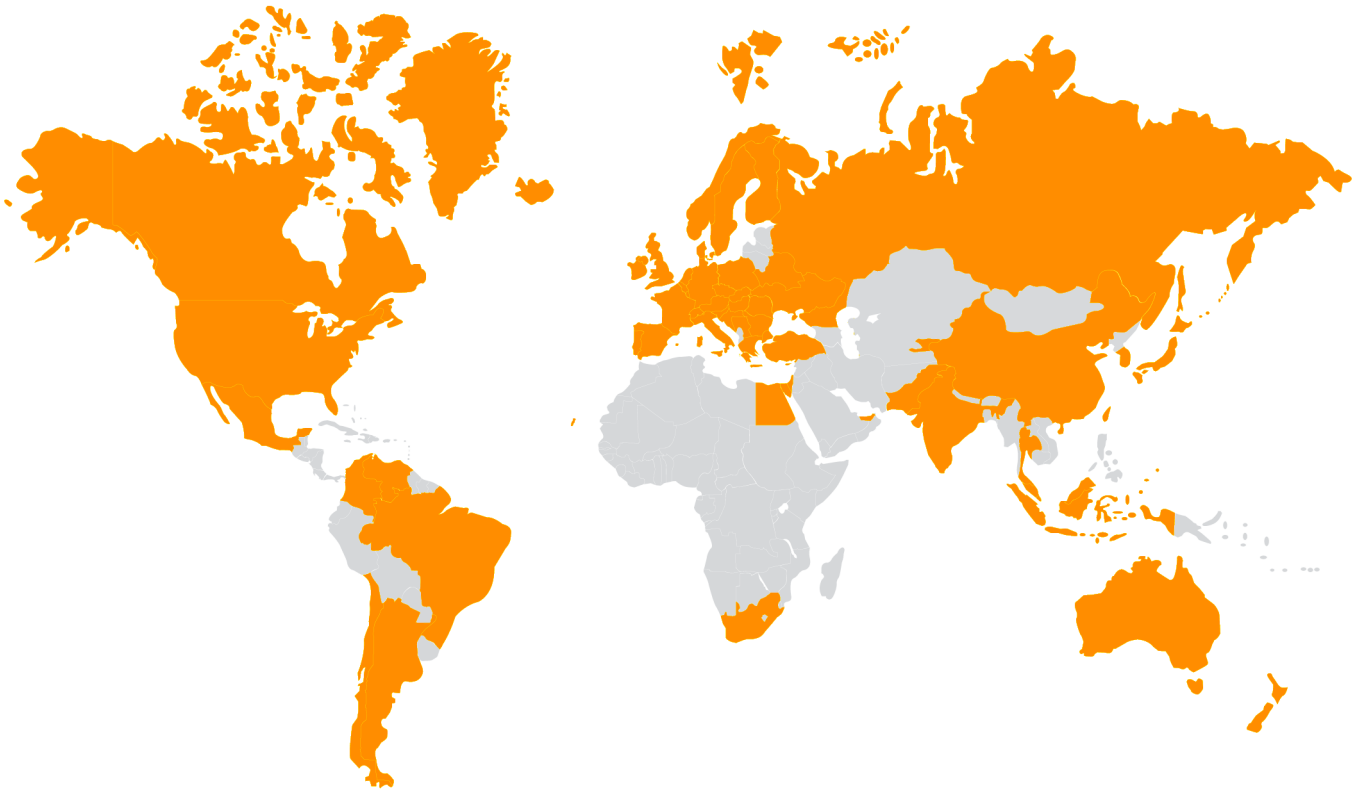
**) see Product Catalog

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Strasse 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

TM260TRE.25-ENG 0907

140 offices in more than 55 countries - **www.br-automation.com/contact**



Australia • Argentina • Austria • Belarus • Belgium • Brazil • Bulgaria • Canada • Chile • China • Colombia • Croatia • Cyprus
Czech Republic • Denmark • Egypt • Emirates • Finland • France • Germany • Greece • Hungary • India • Indonesia
Ireland • Israel • Italy • Japan • Korea • Luxemburg • Kyrgyzstan • Malaysia • Mexico • The Netherlands • New Zealand
Norway • Pakistan • Poland • Portugal • Romania • Russia • Serbia • Singapore • Slovakia • Slovenia • South Africa
Spain • Sweden • Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • USA • Venezuela • Vietnam