TM250

Memory management and data processing

PERFECTION IN AUTOMATION
www.br-automation.com

Prerequisites and requirements

Training modules	TM213 - Automation Runtime TM246 - Structured Text (ST)
Software	Automation Runtime 4.25 Automation Studio 4.2.5
Hardware	ArSim / X20CP1586

Table of contents

1 Introduction

This training module deals with the different possibilities that are available to effectively format, manage and structure data.

One of the most important aspects that will be covered has to do with the correct usage of basic and user-defined data types in the areas of programming, data storage and communication.

Using variables, data types and constants correctly not only helps prevent errors, it also improves the overall flexibility and consistency of the application at hand.

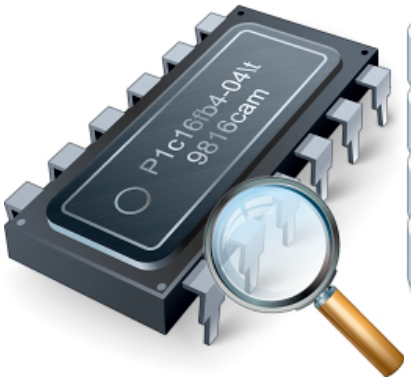


Figure 1: Memory and variables

Figure 2: Data processing and storage

Figure 3: Communication and data exchange

The information presented here gives an impression of the tools and procedures that are available for processing data on the controller.

In addition, it is meant to provide an overview of possible storage formats and locations for data in order to aid in the decision-making process when solving various tasks. Automation Studio library examples and mapp Technology offer a quick way to test functions. Data can easily be saved to files or sent via a network. Automation Studio also allows users to create custom user libraries and provides extensive help documentation to do so.

All of the programming examples included here have been written in the Structured Text programming language. IEC text format has been used for the declarations of variables, data types and constants. In Automation Studio, either the text editor or table editor for the declaration can be used.

1.1 Learning objectives

This training module uses selected application examples and exercises to help participants learn the basics of memory management and data processing on the controller.

- You will learn how to initialize and use variables and data types.
- You will learn how to work with simple and complex data types.
- You will learn about how enumeration and user data types are used.
- You will learn about the various options for managing memory in an application.
- You will learn how to correctly call functions and function blocks from B&R standard libraries.
- You will learn how to create user libraries.

- You will learn the basics of data processing.
- You will learn how to import and use B&R library examples and how to use the integrated Automation Studio help system.
- Participants will receive an overview of data storage and file management.
- Participants will be able to use the mapp Technology recipe system and save recipes on a USB flash drive.
- You will learn about the options available for data transfer and communication using B&R standard libraries.
- Participants will have an overview of the functions for connectivity and access & security and will be able to configure the OPC UA server.

2 Variables, constants and data types

Programming isn't carried out by accessing fixed memory addresses, but via symbolic elements that have names. These elements are called process variables.

Basic data types determine the value range of a variable in addition to how much space it needs in memory. Data types also establish whether values are signed or unsigned, whether they include decimal places, text or even dates or times.

The following sections will provide a brief explanation of numeral systems, basic data types, arrays and user-defined data types.

Variables and data type declarations are displayed in this training module using the IEC text format. Automation Studio users can choose between opening declarations in the table editor or text editor. When double clicking on a declaration file, the table editor is opened by default.

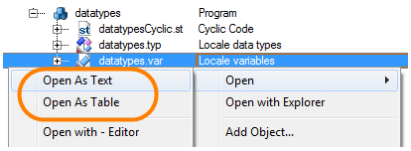


Figure 4: Opening the declaration as a text or table

The declaration will appear as text or table depending on the type of display selected.

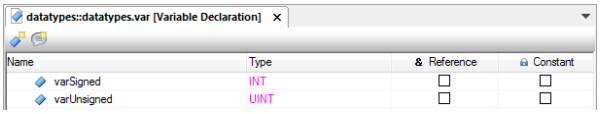


Figure 5: Displaying a declaration as a table

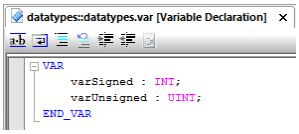


Figure 6: Displaying a declaration as a text

Programming \ Editors \ Table editors \ Declaration editors

2.1 Basic information

At the most basic level, any computer is capable of displaying and calculating values through the interpretation and grouping of the electrical states 0 and 1. It is useful to be familiar with these basic functions.

If you can understand this system, it makes programming a system and localizing errors that may occur much easier.

2.1.1 The binary and hexadecimal systems

A bit is the absolute smallest unit of information and can only take on the states 0 and 1. According to the IEC[1], a bit is regarded as a BOOL value.

Other data types consist of multiple bits that are divisible by eight. The next largest unit is called a byte. A byte is therefore made up of eight bits.

The bits within a byte are numbered from right to left, from **Bit 0** to the most significant value **Bit 7**. Bit 2 (which is actually the 3rd bit) therefore has a decimal value of 4.

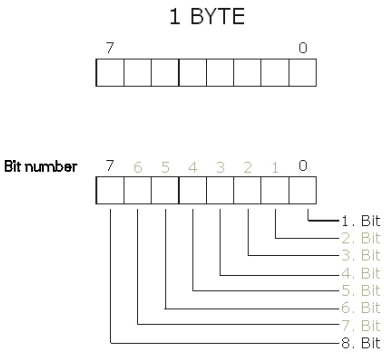


Figure 7: Binary representation of a byte

A byte can be split up into two half-bytes. These half-bytes are often referred to as "nibbles". Logically, the lower nibble is called the low nibble, and the higher nibble is called the high nibble.

The most significant bits are therefore to be found in the high nibble.

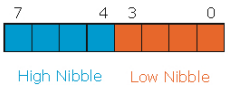


Figure 8: High nibble and low nibble

Each bit within a byte can take on the value 0 or 1. A byte can take on values from 0 to 255, which corresponds to 256 different states.

Bit pattern	Bit number	Decimal value	$2^{\text{Bit number}}$
00000001	0	1	2^0
00000010	1	2	2^1
00000100	2	4	2^2
00001000	3	8	2^3
00010000	4	16	2^4
00100000	5	32	2^5
01000000	6	64	2^6
10000000	7	128	2^7

Table 1: The values of individual bits within a byte

[1] The IEC 61131-3 standard specifies data types, programming languages and file formats that are not dependent on any particular platform.

Two binary numbers will be added together in this example.

+	00000001 One 00000001 One
=	00000010 Two (not decimal 10!)

Table 2: Addition of binary numbers with carry over

The following applies:

0 + 0 = 0

0 + 1 = 1

1 + 1 = 0 + carry over to the next bit

With negative numbers in the binary system, the highest bit is reserved for the sign. As a result, 7 bits are left to represent the value.

Bit 7 is the preceding sign	
X0000000	Bits 0 through 6 for the numeric range
X1111111	The largest positive number is decimal 127

Table 3: Bit pattern for negative integer values

Negative numbers are put together using two's complement. The bit pattern is created from the positive decimal number. This bit pattern is then inverted, and 1 is added. The result then corresponds to the bit pattern for the negative number.

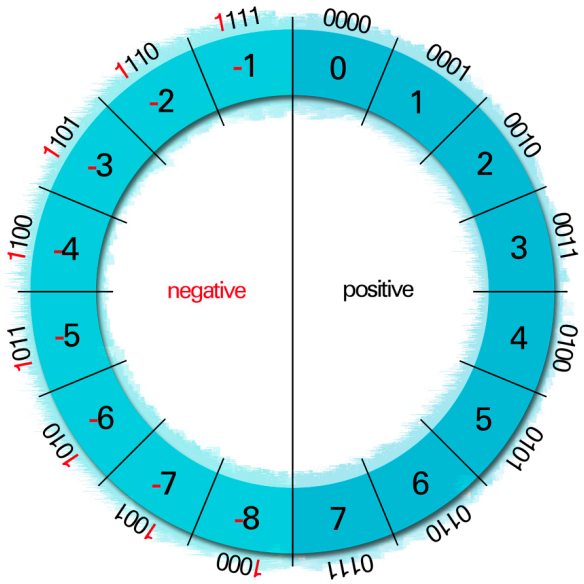


Figure 9: Representation of negative numbers in the binary system

Representation of negative numbers in the binary system

	00000011	Number is decimal "3"
	11111100	Invert all bits
+	00000001	Add 1
=	11111101	Number is decimal "-3"

Table 4: Calculating the two's complement

The correct **representation** of numbers **depends on the data type**.

If a negative value with a signed data type (SINT, INT, DINT) is assigned to an unsigned data type (USINT, UINT, UDINT), then the bit pattern will remain the same. The value will appear differently, however.

This example will illustrate the relationship between data and data types.

Declaration

```
VAR
    varUnsigned : USINT := 0;
    varSigned : SINT := 0;
END_VAR
```

Program code

```
varSigned := -22; (*bit pattern 1110 1010 *)
varUnsigned := varSigned;
```

Table 5: Assigning an unsigned data type to a signed data type

The "varUnsigned" variable is displayed as decimal 234, which corresponds to the bit pattern 1110 1010.

The bit pattern is not changed, but a different value is displayed due to the change in data type (unsigned).

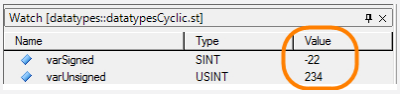


Figure 10: Displaying both variables in decimal number format

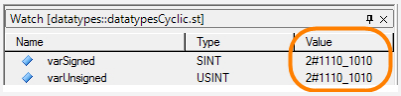


Figure 11: Displaying both variables in binary number format

Programming \ Variables and data types \ Data types \ Primitive data types \ INT (INTEGER)

Programming \ Variables and data types \ Variables \ Bit addressing

In contrast to the decimal system, the hexadecimal system provides 16 values (0 through F) for a single position.

0000	Decimal 0	Hex 0
0001	Decimal 1	Hex 1
0010	Decimal 2	Hex 2
0011	Decimal 3	Hex 3
0100	Decimal 4	Hex 4
0101	Decimal 5	Hex 5
0110	Decimal 6	Hex 6
0111	Decimal 7	Hex 7
1000	Decimal 8	Hex 8
1001	Decimal 9	Hex 9
1010	Decimal 10	Hex A
1011	Decimal 11	Hex B
1100	Decimal 12	Hex C
1101	Decimal 13	Hex D
1110	Decimal 14	Hex E
1111	Decimal 15	Hex F

Table 6: Converting binary numbers to hexadecimal

Nibbles and hexadecimal

0100	1011	Corresponds to 75		(64 + 8 + 2 + 1 = 75)
0100		High nibble	=	4
	1011	Low nibble	=	B
0100	1011	Both nibbles	=	16#4B = 75

Table 7: Converting binary to the hexadecimal system

Nibbles can simply be copied from the binary to the hexadecimal number system and written next to each other.

Memory depth in the binary and hexadecimal systems

Binary				Hexadecimal	Memory depth
00000000				16#00	1 bytes
00000000	00000000			16#0000	2 bytes
00000000	00000000	00000000	00000000	16#00000000	4 bytes

Table 8: Memory depth and representation in binary and hexadecimal

Hexadecimal representation of numbers is primarily used when logbook entries or addresses are displayed.

Localizing errors in signed and unsigned variables is more effective when comparing bit patterns. The Watch window offers the option of displaying variable values in binary, decimal or hexadecimal.

The IEC standard specifies displaying a binary number with the literal 2#0000_1001 and a hexadecimal number with 16#09 in the program code.

Programming \ Standards \ Literals in IEC languages

2.1.2 Comparing variables and constants

Variables are locations in memory that can be changed and take on new values at runtime. Some examples of variables include digital and analog inputs/outputs as well as auxiliary flags.

In contrast to variables, constants are assigned a value when they are being declared. This value can no longer be changed at runtime. Constants are used in the program code as limit values, for example.

The range of values of any of the available basic data types can be used in either case. Overview of IEC 61131-3 base data types

Task: Assigning a value to a constant

1) Declare constants.

   Create the MAX_INDEX variable in the declaration window. It should be of data type USINT. Select the option "Constant" and assign the value 123.

2) Assign a value to the constant.

   In the program code, assign the new value 43 to the constant.

3) Evaluate the output from the compiler.

   Compile the program and analyze the output from the compiler.

Assigning values to constants is not possible in the program code. The compiler will report the following error in the message window:

```
Error 1138: Cannot write to variable 'MAX_INDEX'.
```

Constants can also be used to initialize arrays. In order to do this, however, the project setting "Allow extension of IEC standards" needs to be enabled.

Programming \ Variables and data types \ Variables \ Constants

Project management \ The workspace \ General project settings \ Settings for IEC compliance

2.1.3 Declaring variables, constants and data types

The image shows the Logical View with global and local declaration files.

Variables and data types can be configured in either a table editor or a text editor.

The Automation Studio interface will not be looked at any further in the training module. Information about using the editor can be found in the Automation Studio help documentation.

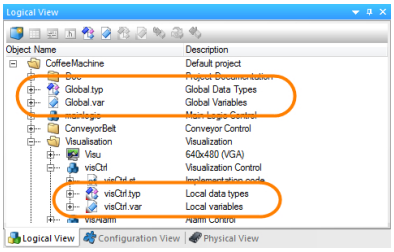


Figure 12: Declaration files in the Logical View

The variable declaration window

Variables and constants are generally only stored in files with the extension .var. A variable declaration file with the name of the program is typically created whenever a new program is inserted into the project.

The data type declaration window

User-defined, enumerated and derived data types are always stored in files with the extension .typ. When a new program is inserted into a project, the user is always prompted to create a data type declaration file as well. It is given the same name as the program.

Programming \ Editors \ Table editors \ Declaration editors \ Variable declaration

Programming \ Editors \ Table editors \ Declaration editors \ Data type declaration

Programming \ Editors \ General operation \ Smart Edit

Initialization

Variables, constants, data types and structure variables can be initialized directly in the declaration editor.

Programming \ Editors \ Table editors \ Declaration editors \ Variable declaration

Programming \ Editors \ Table editors \ Declaration editors \ Data type declaration

Programming \ Editors \ General operation \ Dialog boxes for input support \ Dialog box for initializing structure types and instances

Scope

The scope of declared variables and data types depends on the position of the declaration file in the Logical View.

Programming \ Variables and data types \ Variables \ Scope of declarations

2.2 Primitive data types

Primitive data types are also called basic data types. These data types form the basis for all other derived data types.

2.2.1 Overview of IEC 61131-3 base data types

The following list contains all of the basic data types in line with the IEC 61131-3 standard as well as their areas of use.

Binary / Bit string	Signed integers	Unsigned integers	Floating point	String	Time, date
BOOL	SINT	USINT	REAL	STRING	TIME
BYTE	INT	UINT	LREAL	WSTRING	DATE_AND_TIME (DT)
WORD	DINT	UDINT			DATE
DWORD					TIME_OF_DAY (TOD)

Table 9: Overview of IEC data types

A complete list of primitive data types, their areas of use and range of values can be found in Automation Help.

The data types BYTE, WORD and DWORD are pure bit arrays. Arithmetic operations are therefore not allowed by the compiler.

One feature of IEC data types is that they are completely independent of the platform being used. This means that IEC data types always have the same range of values regardless of the processor architecture or program code where they are being used.

Programming \ Variables and data types \ Data types

2.2.2 Data type conversion

During programming, it may become necessary to convert one data type to another.

When assigning a variable of a data type with a smaller range of values to one with a larger range, implicit conversion is carried out. When the opposite is done (the range of values becomes smaller), the user has to handle the conversion in the program code itself, i.e. explicitly.

The IEC 61131-3 conversion functions are contained in the library AsIecCon, which is automatically part of a new Automation Studio project.

Figure 13: Data type conversion

Implicit data type conversion

Implicit data type conversion occurs when the compiler handles the conversion of one data type to another.

In this program code, a type with a larger range of values takes on the value from a data type with a smaller range.

Declaration	`VAR`
	`  bigValue : DINT := 0;`
	`  smallValue : SINT := 0;`
	`END_VAR`

Program code	`bigValue := smallValue;`

Table 10: Implicit data type conversion

This type of assignment guarantees that the value will have enough space in the new data type. The user doesn't have to perform the conversion himself. The compiler carries out the conversion implicitly.

Explicit data type conversion

If the value a data type with a larger range of values is assigned to a data type with a smaller range, then it's up to the user to carry out the conversion.

Declaration	`VAR`
	`  bigValue : DINT := 0;`
	`  smallValue : SINT := 0;`
	`END_VAR`

Program code	`smallValue := bigValue;`

Table 11: Explicit conversion required

In this case, the compiler will output the following message:

```
Error 1140: Incompatible data types: Cannot convert DINT to SINT.
```

The functions in the "AsIecCon" library can be used to carry out the conversion. This library is included automatically when an Automation Studio project is created.

The conversion in the example above can be carried out properly as shown below.

The expression to be converted is placed inside parentheses with the conversion function directly preceding it.

Declaration	`VAR`
	`  bigValue : DINT := 0;`
	`  smallValue : SINT := 0;`
	`END_VAR`

Program code	`smallValue := DINT_TO_SINT(bigValue);`

Table 12: Carrying out explicit conversion

Programming \ Libraries \ IEC 61131-3 functions \ AsIecCon \ Function blocks and functions

2.3 Arrays, structures and enumeration

User-defined data types can be created that are based on the different primitive data types. This method is called composition. User-defined data types consist of elements from the primitive data types.

These derived data types include the following:

- Arrays and multidimensional arrays
- Structures
- Directly derived types and subranges
- Enumerations

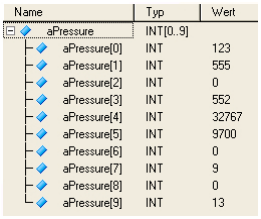
Programming \ Variables and data types \ Data types \ Derived data types

2.3.1 Arrays

Unlike variables of a primitive data type, values of the same data type are outlined in arrays. Each individual element can be accessed using the array's name and an index value.

The value of the array index may not fall outside of the array's actual size. The size of an array is defined when the variable is declared.

In the program, the index can be a fixed value, a variable, a constant or an enumerated element.



Name	Typ	Wert
aPressure	INT[0..9]	
aPressure[0]	INT	123
aPressure[1]	INT	555
aPressure[2]	INT	0
aPressure[3]	INT	552
aPressure[4]	INT	32767
aPressure[5]	INT	9700
aPressure[6]	INT	0
aPressure[7]	INT	9
aPressure[8]	INT	0
aPressure[9]	INT	13

Figure 14: An array of data type INT with a range of 0 to 9 corresponds to 10 different array elements.

Declaring and using arrays

When an array is declared, it must be given a data type and a dimension. The usual convention is for an array's smallest index value to be 0. It is important to note in this case that the maximum index for an array of 10 elements is 9.

Declaration	`VAR`
	`  aPressure : ARRAY[0..9] OF INT := [10(0)];`
	`END_VAR`

Program code	`(*Assigning value 123 to index 0*)` `aPressure[0] := 123;`

Table 13: Declaring an array of 10 elements, starting index = 0

If attempting to access an array element with index 10, the compiler outputs the following error message:

Program code	`aPressure[10] := 75;`
Error message	`Error 1230: The constant value '10' is not in range '0..9'.`

Table 14: Accessing an array index outside of the valid range

If an array of 10 elements should be declared, it can be done in the declaration editor with either "USINT[0..9]" or "USINT[10]"[2]. In both of these cases, an array with a starting index of 0 and a maximum index of 9 will be created.

Task: Creating the "aPressure" array

1) Add new "int_array" program in the Logical View
2) Open the variable declaration window.
3) Declare the "aPressure" array.

   The array should contain 10 elements. The smallest array index is 0. The data type must be INT.
4) Use the array in program code.

   Use the index to access the array in the program code. Use fixed numbers, constants and a variable for this.
5) Force an invalid array access in the program code.

   Access index value 10 of the array and then analyze the output in the message window.

When assigning `aPressure[10] := 123;`, the compiler reports the following error message.

`Error 1230: The constant value '10' is not in range '0..9'.`

The compiler is not able to check array access if the assignment is made using a variable.

```
index := 10;
aPressure[index] := 123;
```

[2] This input method is only supported by the table editor in Automation Studio.

Declaring an array using constants

Since using fixed numeric values in declarations and the program code itself usually leads to programming that is unmanageable and difficult to maintain, it is a much better idea to use numeric constants.

The upper and lower indexes of an array can be defined using these constants. These constants can then be used in the program code to limit the changing array index.

Declaration

```
VAR CONSTANT
    MAX_INDEX : USINT := 9;
END_VAR
VAR
    aPressure : ARRAY[0..MAX_INDEX] OF INT ;
    index : USINT := 0;
END_VAR
```

Program code

```
IF index > MAX_INDEX THEN
    index := MAX_INDEX;
END_IF
aPressure[index] := 75;
```

Table 15: Declaring an array using a constant

The program code has now been updated so that the index used to access the array is limited to the maximum index of the array. An advantage of this is that arrays can be resized (larger or smaller) without having to make a change in the program code.

Programming \ Variables and data types \ Data types \ Derived data types \ Arrays

Task: Calculating the sum and average value

The average value should be calculated from the contents of the "aPressure" array. The program has to be structured in such a way that the least amount of changes to the program are necessary when modifying the size of the array.

1) Calculate the sum using a loop.

   Fixed numeric values may not be used in the program code.

2) Calculating the average value

   The data type of the average value must be the same as the data type of the array (INT).

A constant that is already being used to determine the array size for the declaration of an array variable can also be used as end value for the loop. When generating the average, the same constant can also be used for division. A data type that is larger than the output data type (e.g. DINT) must be used when adding up the individual array elements. The resulting value can then be converted back to the INT data type using an explicit data type conversion.

Multidimensional arrays

Arrays can also be composed of several dimensions. The declaration and usage in this case can look something like this:

Declaration

```
VAR
    Array2Dim : ARRAY [0..6,0..6]  OF INT;
END_VAR
```

Program code

```
(*Zählweise mit 0 beginnend*)
Arry2Dim[2,2] := 11;
```

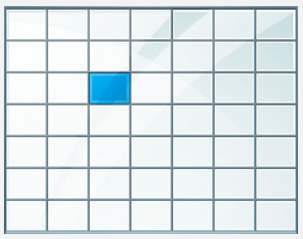


Figure 15: Accessing the value in Column 3, Row 3

Table 16: Declaring and accessing a 7x7 two-dimensional array

An invalid attempt to access an array in the program code using a fixed number, a constant or an enumerated element will be detected and prevented by the compiler.

An invalid attempt to access an array in the program code using a variable cannot be detected by the compiler and may lead to a memory error at runtime. Runtime errors can be avoided by limiting the array index to the valid range.

The IEC Check library can be imported into an Automation Studio project to help locate runtime errors.

Programming \ Libraries \ IEC Check library

2.3.2 Direct composition and subranges

In addition to arrays, other derived data types can also be derived from primitive data types.

It is possible to derive these derived data types directly from the primitive data types. Doing so creates a new data type with a new name that has the same properties as the primitive data type. New data types can also be given an initial value. As a result, all of the variables of this data type have this configured value.

A value range can also be specified for directly derived data types. It is then only possible to assign values to variables of this type that fall within the configured value range. Variables themselves can also be assigned a subrange.

Variable with a subrange	`VAR`
	`    varSubRange : USINT(24..48);`
	`END_VAR`

Data type with a subrange	`TYPE`
	`    Voltage_typ : USINT(12..24);`
	`END_TYPE`

Table 17: Declaring a variable and data type with a subrange

Programming \ Variables and data types \ Data types \ Derived data types \ Directly derived data types

Programming \ Variables and data types \ Data types \ Derived data types \ Subranges

Task: Declaring a directly derived type with a subrange

A new data type "pressure_typ" should be composed. The basic data type should be INT. The valid range of values should fall between 6500 and 29000. Use numeric constants to define this value range.

1) Declaring a directly derived type

   Open up the data type declaration editor and declare the new type by giving it the name "pressure_typ" and assigning "INT" as its basic data type.

2) Declare the constants.

   Open up the variable declaration editor, create the constants "MIN_VAL" and "MAX_VAL" and assign them the values given above.

3) Use the constants to define the subrange of "pressure_typ".

   Now use the two constants to set the value range for the new data type.

4) Test your results.

   Declare a new variable of data type "pressure_typ" in the variable declaration editor. Then try to assign a value outside of the valid range of values in the program code. Take a look at the compiler output in the message window.

2.3.3 Structure types

A structure is composed of individual elements, those being primitive data types, arrays and other structures. The entire structure is addressed via a common name. Each of the individual elements also has its own name. Structures are also known as user data types.

Structures are primarily used to group together data and values that have a relationship to each other. An example would be a recipe that always uses the same ingredients, but in different amounts.

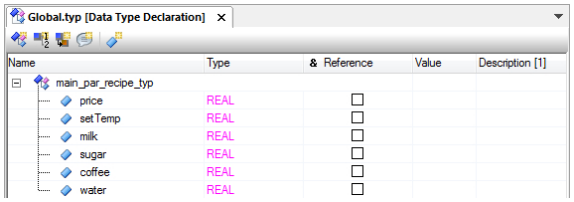


Figure 16: Structure declaration in Automation Studio

Structure declaration	`TYPE`
	`  main_par_recipe_typ : STRUCT`
	`  price : REAL;`
	`  setTemp : REAL;`
	`  milk : REAL;`
	`  sugar : REAL;`
	`  coffee : REAL;`
	`  water : REAL;`
	`  END_STRUCT;`
	`END_TYPE`

Variable declaration	`VAR`
	`  AnyCoffee : main_par_recipe_typ;`
	`END_VAR`

Usage in the program code	`(*price for AnyCoffee*)` `AnyCoffee.price := 1.69;`

Table 18: Declaring a structure, usage in the program code

Programming \ Variables and data types \ Data types \ Derived data types \ Structures

Programming \ Editors \ Table editors \ Declaration editors \ Data type declaration

Task: Declaring the structure "recipe_typ"

Declare a structure with the name "recipe_typ".

This structure should include the following elements:

- price
- milk
- sugar
- coffee
- water

Any data type can be chosen for each element; it depends on how the elements themselves are used in the program.

1) Structure declaration

   Open the declaration editor and create a new structure with the "Add structure type" icon. Assign it the name "recipe_typ". Add the elements according to the list above.

2) Initialize the elements in the program code.

   Declare a new variable with the name "cappuccino" and use the new data type. Use the variable in your program code and initialize the elements with values.

Arrays of structures

Structures can also be declared as arrays. The same rules apply in this case as with arrays of primitive data types. Once again, an index is used for access and must be limited in the application so that memory is not accessed incorrectly.

Declaration	`VAR`
	`    aCoffee : ARRAY[0..5] OF recipe_typ;`
	`END_VAR`

Program code	`aCoffee[0].water := 12;`

Task: Adding together the values of the "sugar" element

Declare an array of "recipe_typ" structures. This array should be dimensioned for 10 elements. Initialize the array with random values.

Use constants instead of fixed values. Add together the "sugar" element from all of the structures and determine how much sugar is needed on average for your production.

1) Declaring the "aCoffee" variable

   Use the "recipe_typ" data type. This array size should be 10 elements.

2) Structure variable initialization

3) Adding together the values of the "sugar" element

   A loop can be used to add up the "sugar" values. Use a constant for limitation of the loop.

4) Calculating the "sugar" average

   After the values have been added up, calculate the average for the "sugar" element. Always test the value change for the first and last array index. This is an easy way to make sure that your program is working properly.

Structure variables can either be written in the source code or in the declaration editors.

Default values for the individual structure elements can be set in the data type declaration. The default values can be overwritten in the variable declaration.

The editor also supports duplicating value lists if the elements of an array have been pre-initialized with the same values.

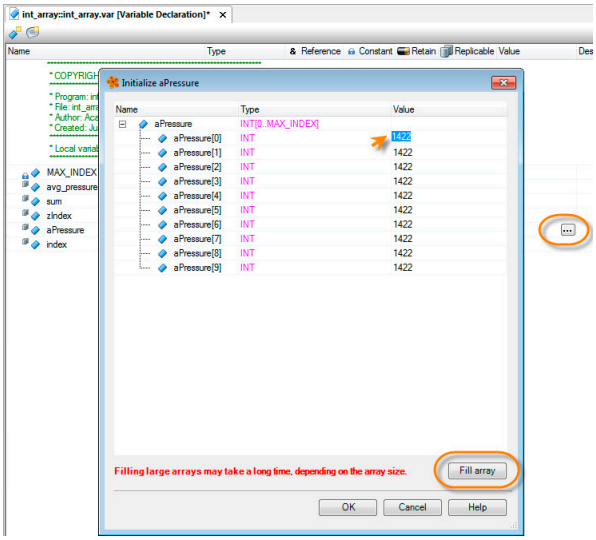


Figure 17: Declaration editor: easy initialization of arrays and structures by clicking "Fill array"

2.3.4 Enumerated data types

Enumeration is basically another way of referring to a list of values. It is therefore possible to use enumerated data types with variables. Instead of the value of the listed element, the variable contains the corresponding text.

The values of the enumerated elements are numbered automatically in ascending order beginning with 0. Initial values can also be assigned.

It often is a good idea to use enumerated types and the elements they contain to indicate the different states of a machine.

Enumerated type declaration

```
TYPE
    Color :
        (
          red,
          yellow,
          green
        );
END_TYPE
```

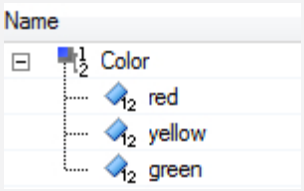


Figure 18: Enumerated type declaration

Variable declarations

```
VAR
    stepColor : Color;
    result : USINT;
END_VAR
```

Program code

```
CASE stepColor OF
    red:
         result := 1;
    yellow:
         result := 2;
    green:
         result := 3;
END_CASE
```

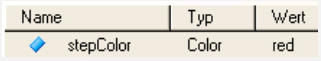


Figure 19: Enumerated type displayed in the variable monitor

Table 19: Declaring an enumerated type, usage in the program code

Programming \ Variables and data types \ Data types \ Derived data types \ Enumeration

2.4 Strings

Strings refer to the sequential arrangement of individual bytes. Each byte contains an alphanumeric character or a control character. When put together, this chain of characters comprises a string.

If a string with 10 characters is defined, then it has 10 usable characters. Every string is terminated with a binary "0", which doesn't count as a usable character. In other words, a string with 10 characters actually takes up 11 bytes in memory.

If only a part of the usable characters is actually used, then the contents of memory following the binary 0 remain undefined.

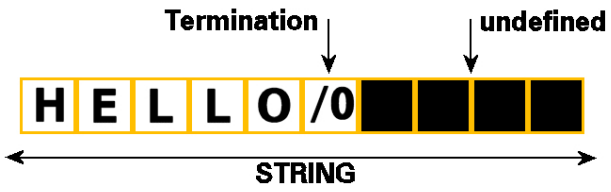


Figure 20: String of 10 characters, zero-termination after "Hello"

2.4.1 Using strings in the program code

The table below shows how the declaration (including initialization) is handled. The number of usable characters is specified inside the brackets. The assignment operator allows a string to take on a text either in the declaration or in the program code itself.

Declaration	`VAR`
	`    sDescription : STRING[80] := 'Test';`
	`END_VAR`

Program code	`sDescription := 'Hello World';`

Table 20: Declaring a string, usage in the program code, assigning a text

If a longer string is assigned to the string variable (i.e. longer than when the string variable was declared), then the string is truncated by the compiler.

Programming \ Variables and data types \ Data types \ Primitive data types \ STRING, WSTRING

Programming \ Standards \ Literals in IEC languages

Programming \ Standards \ ASCII tables

Task: Using a string variable

Declare a string variable called "recipe_typ". It should contain 10 usable characters. Then try to assign it the text "Automation Studio" in the program code and take a look at the results in the variable monitor.

1) Declare the string variable "sDescription".
2) Use it in your program.

   Assign the text "Automation Studio".

   ```
   sDescription := 'Automation Studio';
   ```

3) Check the results in the variable monitor.

   Try to overwrite the text in the variable monitor and observe what happens.

2.4.2 String functions

Strings can be used in several different ways in program code.

The possibilities include the following:

- String comparison
- Conversion to string
- Conversion from string
- String manipulation

String comparison

Strings can be checked against each other for sameness. The result is either equal or unequal.

Declaration	`VAR`
	`    sTextA : STRING[80] := 'Perfection';`
	`    sTextB : STRING[80] := ' in Automation';`
	`    equal : BOOL;`
	`END_VAR`

Program code	`IF sTextA = sTextB THEN`
	`    equal := TRUE;`
	`ELSE`
	`    equal := FALSE;`
	`END_IF`

Table 21: String comparison

Since the two strings are not identical, the result is "`equal := FALSE;`"

Some programming languages do not allow strings to be assigned with an assignment operator or compared like a numeric variable using an IF statement.

In these cases, functions from a string handling library are necessary.

If a closer look at the differences between the two strings is needed, the "brsstrcmp" function from the "AsBrStr" library can be used.

Programming \ Libraries \ Configuration, system information, runtime control \ AsBrStr

Programming \ Libraries \ Configuration, system information, runtime control \ AsBrWStr

Library functions are also necessary for the advanced handling of strings. The next section provides an overview about available libraries and explains where they can be used. Libraries available with functions for handling strings:

- Standard
- AsIecCon
- AsBrStr
- AsBrWStr
- AsString

String conversion

As long as the format is correct, it is possible to convert the contents of a string to a numeric value. The reverse is also possible. These conversion functions are a couple of the many functions included in the AsIecCon library, which is automatically added to every new Automation Studio project.

The contents of the "sPressure" variable should be converted to the numeric value "Pressure".

Declaration	`VAR`
	`  Pressure : REAL;`
	`  sPressure : STRING[80] := '12.34';`
	`END_VAR`

Program code	`Pressure := STRING_TO_REAL(sPressure);`

Table 22: Converting STRING to REAL

Conversion in the reverse direction looks like this:

Declaration	`VAR`
	`  Pressure : REAL := 12.34;`
	`  sPressure : STRING[80];`
	`END_VAR`

Program code	`sPressure := REAL_TO_STRING(Pressure);`

Table 23: Converting REAL to STRING

Programming \ Libraries \ IEC 61131-3 functions \ AsIecCon

String manipulation

Not only can strings be compared with one another or converted to numeric values, it is also possible to join strings together (concatenation), search for partial strings within longer strings, replace text or even place strings at a certain position in another string.

These tasks can all be handled by functions from the "STANDARD" library.

Declaration	`VAR`
	`  sSourceString : STRING[80] := 'Strings in AS';`
	`  sFindString : STRING[80] := 'in';`
	`  position : INT;`
	`END_VAR`

Program code	`position := FIND(sSourceString, sFindString);`

Table 24: Determining the position of a string within another string

The result is 4 because the sought character string was found on the 4th position as the first in the character string.

Programming \ Libraries \ IEC 61131-3 functions \ STANDARD \ Function blocks and functions \ STRING handling functions

Task: Appending strings

Connect two strings to each other. Use the "CONCAT" function from the "STANDARD" library.

1) Declare the variables.

   Declare the variable "sText1" with the initial value "Hello ", variable "sText2" with the initial value "World!" and the variable sResult.

2) Call the "CONCAT" function.

3) Check the results in the variable monitor.

Additional information

Functions from the "STANDARD" library can be used to manipulate strings. The functions always check the available data length so that it is impossible for memory overruns to occur.

The data that can be specified is usually limited to 32 kB, however. String operations that are able to manipulate larger amounts of data are available in the "AsBrStr" and "AsBrWStr" libraries.

When calling functions from the "AsBrStr" library, it's important to note that the system doesn't check if the result string has enough space in the target variable. If the programming contains errors, memory overruns might occur. For this reason, it's important to take a look at the application and make sure that the memory reserved for the target variable is sufficient for the string operation.

The final length of the string can be determined with the "brsstrlen" function from the "AsBrStr" library or "LEN" from the "STANDARD" library.

Programming \ Libraries \ Configuration, system information, runtime control \ AsBrStr

Programming \ Libraries \ Configuration, system information, runtime control \ AsBrWStr

2.5 Addresses, memory size and dynamic variables

Addresses

All variables, constants, arrays and structures that are created in the variable declaration are assigned a memory address by the compiler. This address marks the starting point in the controller's memory of the data.

At runtime, this memory address can no longer be changed; therefore, these variables are referred to as "static variables". The address where a variable is located can be determined with the ADR() function.

A variable's memory address is particularly important when transferring data to functions and function blocks. In these cases, the starting address of the data is passed along to begin processing the data.
4 "Working with libraries" on page 34

Declaration	`VAR`
	`    aCoffee : ARRAY[0..5] OF recipe_type;`
	`    adr_index_0 : UDINT;`
	`END_VAR`

Program code	`adr_index_0 := ADR(aCoffee[0]);`

Table 25: Determining the address of element 0

Although the memory address can be determined, it must never be used in the program code as a fixed value. This is because a new address is assigned by the operating system every time the controller is booted.

Task: Determining addresses

Use the ADR() function to determine the memory address of the "aCoffee" array. The addresses of Index 0 and Index 1 should be ascertained. Calculate the difference between the two addresses and see whether the results are what you expected.

Memory size

Static variables take up a certain amount of memory. This depends on the data types chosen for each of the variables. It is also sometimes necessary to know exactly how much memory is needed. Memory size can be determined using the SIZEOF() function.

For arrays, the total amount of memory is a multiple of the data types being used in the array. The number of array elements can also be calculated.

Declaration

```
VAR
    aCoffee : ARRAY[0..5] OF recipe_type;
    size_complete : UINT;
    size_single : UINT;
    num_elements : UINT;
END_VAR
```

Program code

```
size_complete := SIZEOF(aCoffee);
size_single := SIZEOF(aCoffee[0]);
num_elements := size_complete / size_single;
```

Table 26: Memory required by an array and its elements, number of elements

Task: Determining memory size

Determine the amount of memory used by the entire "aCoffee" array; also determine the size of the element with index 0. Calculate the number of array elements.

Programming \ Libraries \ IEC 61131-3 functions \ OPERATOR \ Function blocks and functions \ Address and length functions \ ADR

Programming \ Libraries \ IEC 61131-3 functions \ OPERATOR \ Function blocks and functions \ Address and length functions \ SIZEOF

Dynamic variables

Dynamic variables can be declared in the variable declaration using the "REFERENCE TO" keyword. The compiler does not assign a separate memory address to a dynamic variable. The keyword "ACCESS" can be used to access any memory address in the program. After this access, the dynamic variable indicates the referenced memory. This data can be both read and overwritten.

In the example, the "index" variable can be used to change which memory address will be accessed. The variable "dynRecipe" then indicates the memory area selected by the "index" variable.

Declaration	`VAR`
	`    aCoffee : ARRAY[0..5] OF recipe_type;`
	`    dynRecipe : REFERENCE TO recipe_type;`
	`    index : UINT;`
	`END_VAR`

Program code	`dynRecipe ACCESS ADR(aCoffee[index]);`

Table 27: Declaration of a dynamic variable, access to a memory address

Programming \ Variables and data types \ Variables \ Dynamic variables

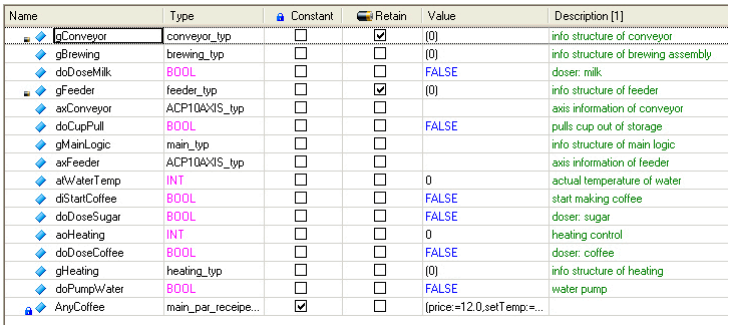
3 Memory management

When designing an application, it is necessary to understand how the system is going to behave. It's important to know how variables, constants, arrays and structures are initialized and what happens to them during booting.

3.1 Memory initialization in the variable declaration

Variables and constants can be initialized when they are declared. If a variable is not assigned an initial value, it always receives the value "0" when the controller boots. Constants must be assigned an initial value.

Initial values for arrays and structures can be assigned in the variable declaration editor. In addition, individual elements of structures can also be assigned with an initial value when the data type is being declared.



Name	Type	Constant	Retain	Value	Description [1]
gConveyor	conveyor_typ	☐	☑	{0}	info structure of conveyor
gBrewing	brewing_typ	☐	☐	{0}	info structure of brewing assembly
doDoseMilk	BOOL	☐	☐	FALSE	doser: milk
gFeeder	feeder_typ	☐	☑	{0}	info structure of feeder
axConveyor	ACP10AXIS_typ	☐	☐		axis information of conveyor
doCupPull	BOOL	☐	☐	FALSE	pulls cup out of storage
gMainLogic	main_typ	☐	☐		info structure of main logic
axFeeder	ACP10AXIS_typ	☐	☐		axis information of feeder
atWaterTemp	INT	☐	☐	0	actual temperature of water
diStartCoffee	BOOL	☐	☐	FALSE	start making coffee
doDoseSugar	BOOL	☐	☐	FALSE	doser: sugar
aoHeating	INT	☐	☐	0	heating control
doDoseCoffee	BOOL	☐	☐	FALSE	doser: coffee
gHeating	heating_typ	☐	☐	{0}	info structure of heating
doPumpWater	BOOL	☐	☐	FALSE	water pump
AnyCoffee	main_par_receipe...	☑	☐	{price:=12.0,setTemp:=...	

Figure 21: Variable declaration window with variables, constants, arrays and structures

All variables and constants are located in the controller's DRAM. If programming is done incorrectly, e.g. by accessing an array index outside of the valid range, it's even possible to manipulate the values of constants.

If variable values should remain after the controller has been restarted, then the "RETAIN" option can be selected during the declaration. Data is then stored on the controller's battery-buffered RAM during a power failure or restart. More information about this can be found in "TM213 - Automation Runtime".

When creating an application, it's important to consider how variables are affected in memory when power failures or restarts occur.

The application must be able to boot with the correct parameters after any kind of initialization or restart.

Programming \ Editors \ Table editors \ Declaration editors

Programming \ Editors \ General operation \ Dialog boxes for supporting input

Programming \ Variables and data types \ Variables \ Variable remanence

Real-time operating system \ Method of operation \ Module / data security \ Power-off handling

Real-time operating system \ Method of operation \ Module / data security \ Power-on handling

Programming \ Libraries \ IEC Check library

3.2 Copying and initializing memory in the program code

Copies memory areas

Depending on the programming language and environment being used, there are several different methods available for copying data. In the Structured Text programming language, data from Variable A can be copied to Variable B through assignment. For this to work, however, the source and target data types must be identical; otherwise, a compiler error results.

If data from one data types needs to be copied to a different data type, this is possible using the "brsmemcpy()" function. The addresses of both the source memory and target memory as well as the number of bytes to be copied need to be specified in this case.

Declaration	`VAR`
	`    aTarget : ARRAY[0..4] OF USINT;`
	`    aSource : ARRAY[0..4] OF USINT;`
	`END_VAR`

Program code	`brsmemcpy(ADR(aTarget), ADR(aSource), SIZEOF(aTarget));`

Table 28: Copying a portion of memory with brsmemcpy()

It is important to make sure that the target memory is large enough for the data block being copied. The functions SIZEOF() and MIN() can be used to determine the size of the smallest memory area. This ensures that only the number of bytes that have space in the target memory are copied over.

Declaration	`VAR`
	`    aTarget : ARRAY[0..2] OF USINT;`
	`    aSource : ARRAY[0..4] OF USINT;`
	`    min_len : USINT := 0;`
	`END_VAR`

Program code	`min_len := MIN(SIZEOF(aTarget), SIZEOF(aSource));` `brsmemcpy(ADR(aTarget), ADR(aSource), min_len);`

Table 29: Copying limited memory areas with "MIN()"

Initializing memory areas

Memory areas with a different structure and data type can be initialized when the variable is being declared. It is sometimes necessary to overwrite certain data areas in the program. This is possible in the program code using the brsmemset() function.

Declaration	`VAR`
	`    aTarget : ARRAY[0..4] OF USINT;`
	`END_VAR`

Program code	`brsmemset(ADR(aTarget), 0 , SIZEOF(aTarget));`

Table 30: Initializing a memory area with "brsmemset()"

Programming \ Libraries \ Configuration, system information, runtime control \ AsBrStr

Task: Initializing and copying memory

1) Declare two variables

   Both variables are to be of user-defined data type "recipe_typ".

2) Initialize the first structure with the brsmemset() function.

   Initialization should not be done cyclically, but by using a command. The initial value should be 0.

3) Copy the data.

   Copy the contents of the first structure to the second structure. Use the brsmemcpy() function for this. Pay attention to the length of the data when using these functions. The copy procedure should also be performed just once by setting a command.

3.3 Reserving memory blocks

In some rare cases, it may be necessary to reserve a memory area in DRAM at runtime. This memory is then available to the user's program. It can be accessed via the reserved memory's starting address. Unless it has been freed up again by the user, this reserved memory area is not available to the system.

The memory being reserved must comprise consecutive available memory locations on the system in order for it to be used by the user program.

The contents of this memory must be initialized by the user. Since the contents of memory reserved are in DRAM, it will be lost each time the controller is restarted.

Memory management with SYS_Lib

If the amount of memory to be reserved is known before the controller is booted, then the "tmp_alloc" and "tmp_free" functions in the SYS_Lib library can be used.

These functions may only be called in the INIT and EXIT subroutines of the controller program.

Programming \ Libraries \ Configuration, system information, runtime control \ SYS_Lib \ Functions and function blocks \ Memory management

Memory management with AsMem

The AsMem library can be used in the INIT subroutine of a program to reserve a large partition of memory on the system. Memory blocks can then be diverted and then freed up using function blocks called in the cyclic program itself.

Programming \ Libraries \ Configuration, system information, runtime control \ AsMem \ Functions and function blocks

Programming \ Examples \ Libraries \ Configuration, system information, runtime control \ Managing memory areas

4 Working with libraries

Libraries allow software to be packaged in compact units and reused whenever necessary. A library is a collection of functions and function blocks, constants and data types.

This section will provide a brief introduction to the terminology of libraries as well as information about how to use functions and function blocks correctly.

Later on, we will also discuss how to create user libraries, which make it possible to store different functions that the user has already created and implemented. The use of sample programs from B&R standard libraries will also be demonstrated.

Figure 22: Working with libraries

4.1 General information

A library is a collection of functions, function blocks, constants and data types. Any existing library can be inserted into the Logical View at any time. Either B&R standard libraries or libraries created by the user can be selected.

Project management \ The workspace \ Catalogs

Function

A function consists of the actual function call, the parameters being transferred and a value that is returned. When a function is called, the parameters are passed and a return value is returned immediately. Different parameters can be transferred the next time the function is invoked.

Program code

```
result := MIN(100,200);
```

Table 31: Calling a function with two parameters

Function block

In contrast to a function, a function block can return more than one value. It is also necessary to declare an "instance". In addition, it is possible for a function block to perform a task by calling it several times.

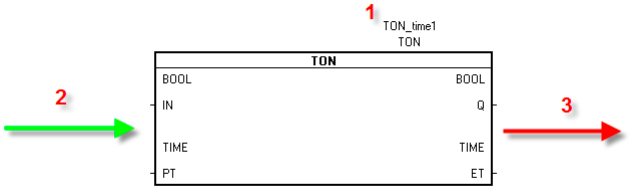


Figure 23: (1) Instance variable, (2) Inputs and (3) Outputs on a function block

1) Instance structure, must be declared in the variable declaration editor with a unique name.
2) Input parameters are passed directly before or during the function block call.
3) Output parameters are written while the function block is being called and can then be used in the program code.

Different instances make it possible to make calculations in tasks using different parameters.

An instance can actually be thought of as a structure. The function block takes in the input parameters at the moment it is called and then passes the output parameters on to the instance.

Declaration	`VAR`
	`    TON_time1 : TON;`
	`    TON_time2 : TON;`
	`END_VAR`

Program code	`TON_time1(IN := enable1, PT := T#5s);` `TON_time2(IN := enable2, PT := T#10s);`

Table 32: Calling two TON function blocks with different times

The two timers can be started at different points in time and run for different periods of time as well. The time that has already passed and the delayed output signal are stored in the instance when the function block is called.

A function block only takes on input parameters when called the next time. The outputs of the function block are only handled when the function block is called.

If the function block name is highlighted, the Automation Help can be called by pressing the F1 key. It contains information about the parameters as well as the inputs and outputs for that particular function or function block.

Task: Generating a clock signal

Call the "TON" function block. Generate a clock signal with a length of one second and a one second pause.

Take a look at the timing diagram in the function block description. This requires that the function block receive the value "FALSE" on the "IN" input for at least one call in order to reset the output "Q".

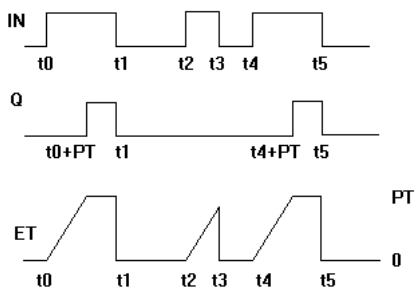


Figure 24: Time diagram of the TON function block from the Automation Studio help system

Programming \ Libraries \ IEC 61131-3 functions \ STANDARD \ Function blocks and functions

Programming \ Functions and function blocks \ Functions

Programming \ Functions and function blocks \ Function blocks

4.1.1 Function blocks with enable input and status output

Many function blocks included in B&R standard libraries have an enable input and a status output.

The enable input makes it possible to enable or disable the function block.

The status output indicates the current status of the function block. There are some statuses that are the same for all of the function blocks. Other statuses can vary, but their number range can always be used to determine clearly the library to which they belong. The status values are provided by the library as constants Information about status values can be found in Automation Help.

Constant	Error number	Function
ERR_OK	0	Execution successful, no errors, output values are valid
ERR_FUB_ENABLE_FALSE	65534	Enable not set, function block called but not executed
ERR_FUB_BUSY	65535	BUSY, function block called but action not yet finished, call again in the next cycle

Table 33: Universal status values of function blocks

Universal status values are not errors and must be handled accordingly when the return values are evaluated in the program code.

These three status values are predefined in the **"Runtime"** library. This library is always included in an Automation Studio project.

Diagnostics & service \ Error numbers \ Libraries \ Runtime

Diagnostics and service \ Error numbers \ AR system \ 65534 - 65535 Function block status

This example shows how to call a function block that has a status output. Correctly evaluating the status output is essential here. The function block being called is "CfgGetIPAddr" from the "AsARCfg" library.

Declaration

```
VAR
    GetIP : CfgGetIPAddr;
    sIPResult : STRING[20];
END_VAR
```

Program code

```
GetIP.enable := TRUE;
GetIP.pDevice := ADR('IF3');
GetIP.pIPAddr := ADR(sIPResult);
GetIP.Len := SIZEOF(sIPResult);
GetIP();
IF GetIP.status <> ERR_FUB_BUSY THEN
    IF GetIP.status = 0 THEN
    (*everything ok*)
    ELSE
    (*place error handling here ... *)
    END_IF
END_IF
```

Table 34: Evaluating the status after a function block call.

It's important to keep calling the function block as long as the status remains equal to BUSY. As soon as the status is equal to 0, the procedure has been carried out correctly and the output parameters can be used in the program. Any status other than 0 or BUSY must be handled in the program code accordingly. An overview of error numbers can be found in the documentation for the respective library.

Exercise: Call a function block with status output

1) Call the "CfgGetIPAddr" function block.

   Call the "CfgGetIPAddr" function block from the "AsARCfg" library. Use it to determine the IP address of the Ethernet interface on your controller. The function block needs the name of the respective interface. This name can be read from the configuration of the respective interface in the Physical View.

2) Evaluate the status values.

   Evaluate the status outputs correctly. Set a variable with the name "status_signal" to 1 if the status = ERR_FUB_BUSY. Set this variable to 2 if the status = 0. If a different error (status) occurs, set the variable to 100.

3) Look up the status values in the help documentation.

   Look for the error codes for this function block in Automation Help. Look up which suggestions the help system offers for correcting errors.

4) How can this error be handled?

Think about how the program might be able to handle this error. Is there any way to implement countermeasures against it in the program code?

Programming \ Libraries \ Configuration, system information, runtime control \ AsARCfg

4.1.2 Function blocks with Enable or Execute and statusID

Many function blocks in the B&R system have an Enable or Execute input and several status outputs.

The function block is enabled and disabled using the enable input.

The respective function is called once using a rising edge on the Execute input. The current state of the function block can then be read via the "Busy", "Done" and "Error" status outputs.

If the "Error" output is set to TRUE, then an extra piece of information is made available on the output "StatusID". The value on the StatusID output, which is also called EventID, is recorded in the help documentation of the respective library.

Programming \ Libraries \ Configuration, system \ ArEventLog \ Function blocks and functions \ EventID functions

4.1.3 Components of a library

A library consists of several components and properties.

These components include the following:

- .fun file: Contains the interface or structure of a function or function block instance
- .var file: Contains numeric constants, which includes the statuses that a function block can return as well as the parameters expected by the function or function block
- .typ file: Structures that are needed internally by the function block or that must be passed to the function block in the application

Figure 25: Components of a library

Programming \ Libraries

Project management \ Logical View \ Libraries

4.2 Creating user libraries

In order to be able to reuse program code, it must first be split up into self-contained modules that can be maintained. User libraries created in Automation Studio are an ideal way to do this.

Before the design phase begins, it is important to give some thought to the scope of the individual functional units, i.e. the functions and function blocks. Once this has been done, the interfaces can be declared. Finally, the range of functions can be implemented.

Figure 26: New user library

The following questions must be taken into consideration when creating a library:

- What is the function of this library?
- Which functions and function blocks are necessary?
- How should the interfaces for the functional units look?
- Will constants and structures be used?
- Are certain things necessary from other libraries to handle certain tasks?
- How will the library be passed on or stored?

Inserting a user library

A new library can be added via the toolbox. Some categories are left over when setting the filter. For generating an IEC library, a corresponding entry from the results list is selected and added to the Logical View.

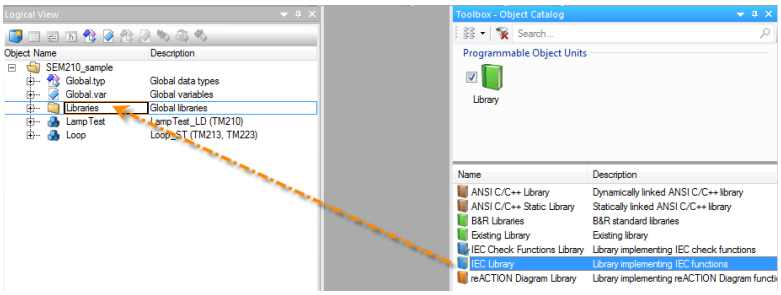
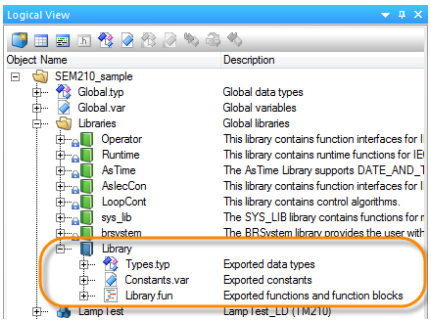


Figure 27: Adding a new IEC library via the toolbox



If the new IEC library has been added to the Logical View, then the name of the library can be changed.

The new library is comprised of the following components:

- Name and description of the library
- Content from the constant declarations (*.var)
- Data type declaration file (.typ)
- Function and function block declaration file (.fun)

Figure 28: Components of a new IEC library

Project management \ Logical View \ Libraries

Adding a function block

After it has been created, the library can be selected in the Logical View. The toolbox can be used to add a function or function block to the new IEC library. When this is done, a wizard appears that can be used to specify the function name, icons and the function block interface.

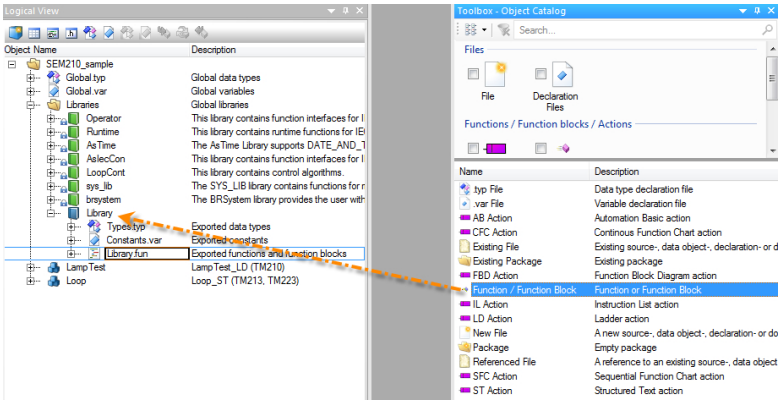


Figure 29: Adding a function block to the new library

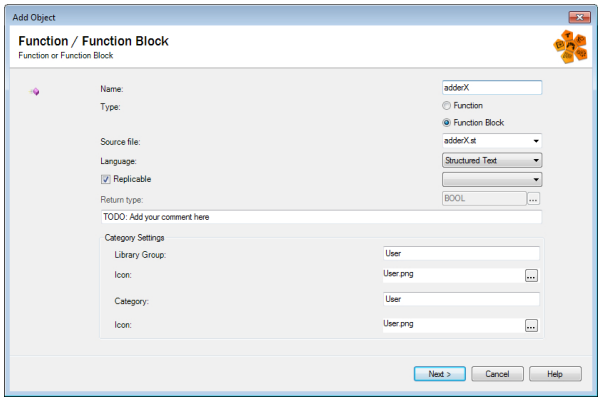


Figure 30: Selecting the category Function / Function block

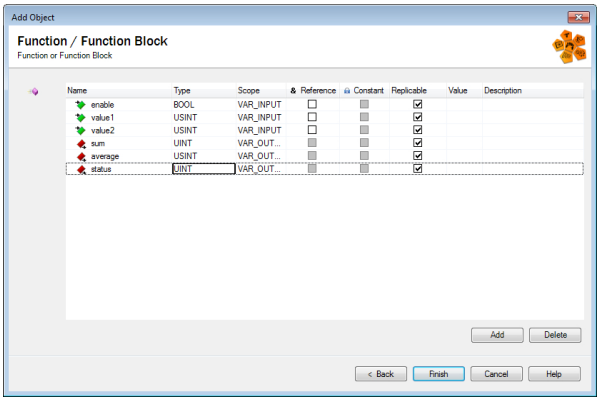


Figure 31: Function block interface configuration

The following settings are possible:

- Name and description of the function or function block
- Selection of whether this is a function or function block
- Programming language of the function or function block
- For functions: Data type of the return value
- Declaration of input and output parameters

After a function or function block has been added, the contents of the .fun file are changed accordingly. Changes can be made to the .fun file at any time.

In addition, a source file with the name of the function or function block is inserted into the library; this is where the actual functionality is implemented.

It is now possible to begin this implementation in the source file. The parameters of the function block interface can be used as variables.

Using StatusID, generating EventIDs

If a function block outputs a StatusID, we recommend that constants be defined for it. These constants can be declared in the library's .var file. Alternatively, the function ArEventLogMakeEventID() from the library ArEventLog is used to generate an EventID. When using this function, a bit is automatically used for labeling user EventIDs. This way it can be ruled out that the system's EventID overlaps with the user's EventIDs.

Programming \ Libraries \ Configuration, system \ ArEventLog \ Function blocks and functions \ EventID functions

The Automation Studio help system provides comprehensive instructions for creating a user library.

Programming \ Libraries \ Example for creating a user library

Implementation and testing

To ensure that the new library functions properly, it must be thoroughly tested. To do so, new functions and function blocks can be called and tested in any program.

Using function blocks

If function blocks from other libraries should be used in the user library, an instance must first be declared. The instance for external function blocks should be declared as an internal variable in the user function block instance.

Exporting and transferring

Once the library is finished, it can be assigned a version number, e.g. in the library's properties. It is then possible to export the user library. This can be done using the File menu in the Logical View. The user can also choose whether the source files should be included.

If the source files are not included, it's important to note that the library can no longer be edited at a later time.

Libraries can be managed in packages in the Logical View, just like programs. The libraries are stored in the same package as the program that is using it.

Task: Creating the library "myMath"

Create the user library "myMath". A function block called "adderx" is needed that will add up the weights on a scale and determine their average value. In addition, the maximum weight should also be determined. The function block must have an Execute input and the status outputs "StatusID" and "Done". If one of the weights has the value "0", then the "Error" output will be set and a corresponding EventID output.

A list of the function block inputs and outputs is provided in the following table.

Name	Properties / Value	Data type
	Parameter (.fun)	
Execute	VAR_INPUT	BOOL
weight1	VAR_INPUT	INT
weight2	VAR_INPUT	INT
weight3	VAR_INPUT	INT
Done	VAR_OUTPUT	BOOL
Error	VAR_OUTPUT	BOOL
StatusID	VAR_OUTPUT	DINT
sum	VAR_OUTPUT	DINT
average	VAR_OUTPUT	INT
Maximum	VAR_OUTPUT	INT

Table 35: Interface and constants for "adderx"

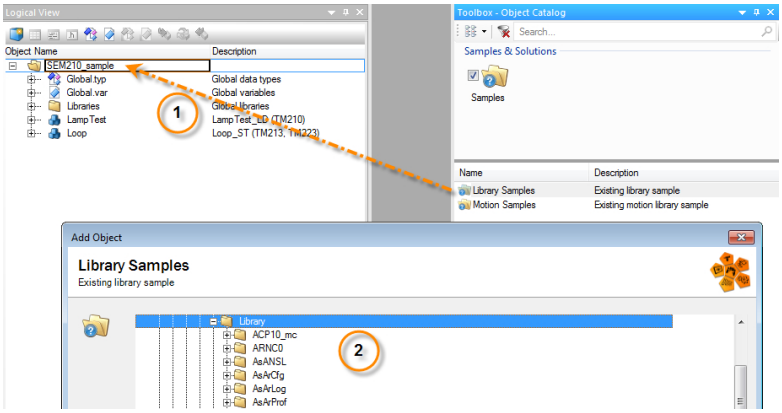
Programming \ Libraries \ Example for creating a user library

Project management \ Logical View \ Libraries \ Exporting a user library

Programming \ Editors \ Table editors \ Declaration editors \ Function and function block declarations

4.3 B&R standard library examples

The use of B&R standard libraries provides comprehensive support when implementing tasks. To make it easier to work with these libraries, B&R has created several library examples. With the help of standard solutions, functions in these libraries can be used even more efficiently.



These ready-made units can be imported using the toolbox in the Logical View.

1) Selection of example types
2) Specifies of the example - Sorting according to libraries

Figure 32: Adding library example from the toolbox

The examples can then be modified and tailored to the specific application (e.g. adjusting interface parameters) before being transferred to the target system at hand.

It was made sure that all the examples can also be used in the simulation (ArSim).

The structure of the individual programs is the same. This provides a quick overview of how libraries can be used in different areas. A description and overview of the examples can be found in Automation Help. The area for the examples has the same layout as the reference documentation for the actual library.

Programming \ Examples \ Libraries

Programming \ Examples \ Adding examples

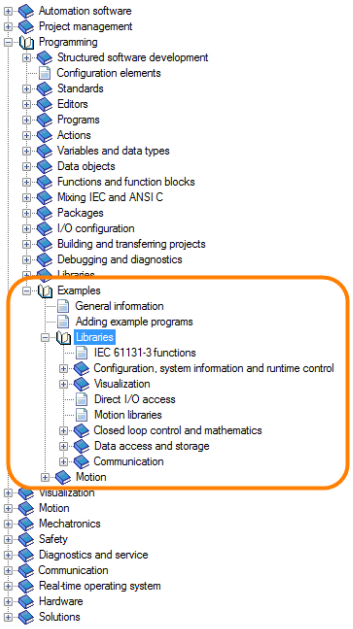


Figure 33: Overview of documentation for examples in the Automation Studio help documentation

Exercise: Add a library example

In the following task we will be adding and starting up a library example in the Logical View.

1) Add the following example in the Logical View:

   Programming \ Examples \ Libraries \ Configuration, system information, runtime control \ Create and evaluate user logbook

2) Assign to software configuration

   When adding the library example, the user is asked whether the programs should be applied to the respective controller's software configuration. If this message is confirmed, the example program is added to the default task class. The assignment can later be changed manually by dragging and dropping in the software configuration.

3) Transfer the program

   Transfer the program to the controller. Depending on the example program, it may be necessary to adjust some parameters in the INIT program. Any such parameters will always be described in the example program's documentation.

4) Test the program

The Automation Studio help system documents all of the parameters and commands used to run the example program. The variable monitor can be used to run the example program by setting commands manually. Status values are provided for monitoring progress.

5 The basics of data processing

Up until now, all variables, arrays and structures have been used locally. It is frequently necessary, however, to store data in a file at a certain location in memory or to transfer it over a network. When doing so, a few things need to be taken into account.

It's important to know how data is stored in memory as well as the format in which it is stored and transferred. In addition, the contents of the data must remain consistent whenever it is being saved or transferred.

5.1 Alignment

Different process architectures follow different rules when it comes to data storage. The user usually doesn't have to take this into consideration, except for in extremely rare cases. If data is to be transferred between systems that have different architectures, however, it is a good idea to think about how that data is to be stored.

Exercise: Create a user-defined data type and determine the memory size

1) Modify the user-defined data types

   Change the element data types for "recipe_typ" according to the following table.

2) Determine memory size

   Declare a variable "SimpleCoffee" that uses the user data type "recipe_typ". Use the sizeof() function to determine the memory size of the structure.

3) Analyze the results

   It is possible that the data length you've determined does not correspond to what you expected.

Element name	Data type
price	USINT
milk	UINT
sugar	USINT
coffee	UDINT

Table 36: Elements of the data type "recipe_typ"

Did the data length determined in the previous task correspond to your expectations? The additional data is in the form of stuff bytes. These are automatically added by the compiler due to the alignment of this particular processor architecture. This is necessary primarily because the processor is able to address the stored data more easily and more quickly. In addition, it is not possible for the processor to read data types with an even data length from memory addresses with an odd length.

Exercise: Modifying the user data type

1) Rearrange the elements.

   Re-sort the elements of the "recipe_typ". In this way, it's possible to use a portion of the stuff bytes for the data in the structure.

5.2 Data formats

Data can be stored and transferred in many different formats. When looking at a structure, for example, its individual bytes are interpreted and represented according to the data types being used.

If the data contents of the structure are copied to a byte field, then only the binary data is left over. In this case, the data can only be interpreted if the storage format, i.e. the user data type in this case, is known.

If data is then stored in files or transferred over a network, it is necessary to know the data format being used on both sides, i.e. where the data is stored or being sent from as well as where the data is ultimately to be interpreted.

There are many different data formats available. Without an appropriate data format, the data still exists in binary form.

Figure 34: Is the data format known?

	Formatted data	Binary data
BINARY	2B 34 33 37 37 34 38 36 35 38 36	2B 34 33 37 37 34 38 36 35 38 36
Data type REAL	12.34	41 43 D7 0A
ASCII	"Hello World!"	48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00
XML	<?xml version="1.0"> <ComboBox> <Item ID="off"/> </ComboBox>	3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E 30 22 3E 0D 0A 3C 43 6F 6D 62 6F 42 6F 78 3E 0D 0A 3C 49 74 65 6D 20 49 44 3D 22 6F 66 66 22 2F 3E 0D 0A 3C 2F 43 6F 6D 62 6F 42 6F 78 3E 0D 0A

Table 37: Comparison of formatted data and binary representation

As can be seen in the table, all data is always stored in binary form. It can only be depicted differently if the data format being used is known. A file extension says nothing about the contents of a file; it simply provides a clue about how the data should be interpreted.

5.3 Data consistency

If the contents of memory are copied from A to B through assignment or with "brsmemcpy()", then this is done immediately with a function call. If the same data should be saved to a file or transferred over a network, then it is necessary for the data to remain consistent.

Saving or transferring data can easily take several program cycles, and it's not possible to predict how much of it can be processed per cycle. For this reason, it is important that the contents not change at all throughout this process.

Before the save or copy procedure begins, it is therefore a very good idea to implement a "preparation step" in the program that gets the data ready. This preparation step is only executed once before the save or transfer procedure is carried out. Data consistency can be ensured in this way.

If data does not remain consistent when being saved or transferred, it could possibly be stored or transferred with data with which it doesn't belong.

For example, a series of measurement values may no longer fit together chronologically. Depending on the process, the effects of this can be minimal or extremely critical.

Declaration

```
VAR
  stepSave : UINT;
  aSend : USINT[0..9];
  aRaw: USINT[0..9];
  cmdSave : BOOL := FALSE;
END_VAR
```

Program

```
CASE sSave OF
  0: (*--- wait for instruction*)
    IF cmdSave = TRUE THEN
      sSave := 1; (*--> Prepare & Save*)
    END_IF
  1: (*--- prepare data*)
    brsmemcpy(ADR(aSend), ADR(aRaw), SIZEOF(aRaw));
    sSave := 2;
  2: (*--- save data into file*)
    SaveData(ADR(aSend));
END_CASE
```

Table 38: Preparing data before saving it to a file

Immediately after "cmdSave" is called, the modified "aRaw" data is copied from the application to a byte field ("aSend"). This remains the same throughout the entire save procedure[3] which ensures data consistency.

[3] The functions called here are symbolic and only used to explain the sequence; they do not actually exist.

6 Storing and managing data

Data can be managed on the controller in many different ways. They can be saved on RAM as a memory block, on process variables, on the Technology Guard or as a file on a file system.

6.1 Data storage on the Technology Guard

Retain variables and permanent variables allow variable values to be stored in buffered memory (battery-backed SRAM or FRAM). In this way, important variable values are still available after restarting the system. However, the saved data is always connected to the respective controller and, for example, can be backed up and restored using the Runtime Utility Center variable services.

The Technology Guard can be implemented as an alternative to data storage on the controller. This, in addition to the license management, offers two manipulation-proof operating hour counters and a 200 byte data storage. These functions are made accessible via the function blocks of the AsGuard library. The Technology Guard can be inserted into another controller if needed, where the data is also included.

The responsibility for managing data on the Technology Guard regarding data consistency and alignment lies with the user.

Figure 35: Technology Guard as data storage

Programming \ Variables and data types \ Variables \

- Variable remanence
- Permanent variables

Automation software \ Technology Guarding

Programming \ Libraries \ Configuration, system \ AsGuard

Programming \ Examples \ Configuration, system information, runtime control \ Technology Guard examples

Task: Data storage on the Technology Guard

The objective of this task is to store the data from the previously set up recipe structure in a system-independent manner. The memory on the Technology Guard is used as the storage location. The task should be tackled based on the library example for the AsGuard library. A few changes to the example should be made for this purpose.

1) Import an example for the AsGuard library
2) Transfer the example to the controller
3) Check if the inserted Technology Guard can be read

4) Transfer the recipe structure as a data source for storing on the Technology Guard
5) Storing the data on the Technology Guard
6) Reading the data from the Technology Guard
7) [additional task]

   If there is time, check if the data from the Technology Guard can be read by the application on a different controller.

6.2 Storing files in the file system

It's not always the case that data remains on the same system. Files are one way to store and transport data at the same time. The libraries FileIO, AsUSB and AsZip make it possible to access the file system on Flash memory, USB flash drives or network resources.

This results in the following:

- Access to the controller's file system
- Management of directories
- Searching of directories
- Creating, reading from and writing to files
- Access to USB mass memory devices
- Access to resources freed up on the network
- Access to the FTP server on the network
- Compress and extract files

Figure 36: Possibilities with the FileIO library

Accessing the file system requires a symbolic name behind which the file destination is configured. This is referred to as a file device. A file device is a name (e.g. "RECIPES" that points to a specific location (e.g. "F:\Recipes\").

File devices can be created in the Automation Studio system configuration.

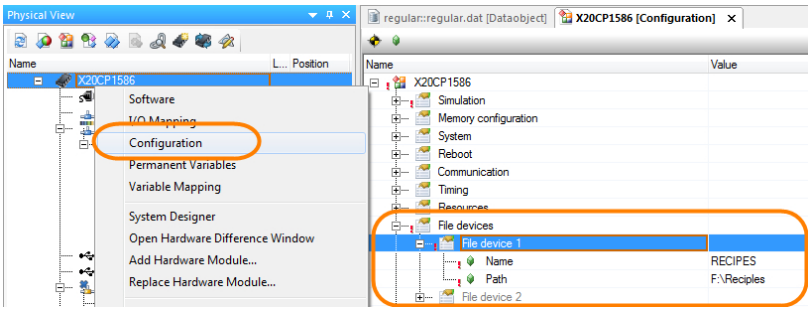


Figure 37: Configuration of file devices in the system configuration

The functions of the AsUSB library are used to recognize USB devices. A file device is set up at operating time using the function block "DevLink()" from the FileIO library.

Programming \ Libraries \ Data access and data storage

- FileIO
- AsZip
- AsUSB

Programming \ Libraries \ Data access and data storage \ FileIO \ General information \ File devices, files and directories

Programming \ Examples \ Libraries \ Data access and data storage

- Data storage
- Data access

When storing data on the Flash memory, it's important that it is stored on the user partition. On normal file systems, this is the "D:\" drive and on secure file systems, this is the "F:\" drive. Using different partitions allows the operating system, application and user data to be clearly delineated.

The Flash memory is partitioned during online installation, offline installation or remote installation.

Project management \ Project installation

Real-time operating system \ Target systems \ SG4 \ AR remote install

Exercise: Determine the content of a USB flash drive

Once a USB flash drive is inserted into the controller, the controller must determine its content. Functions from the AsUSB library are used to recognize the USB devices. In order to generate the FileDevice and read it, the functions from the FileIO library are required. The library example "LibAsUSB1_ST" is used as a basis for this exercise.

1) Import the library example "LibAsUSB1_ST"
2) Transfer the project
3) Insert the USB flash drive into the controller
4) Set variable "start_reading_usb_data" to TRUE
5) Observe behavior of variable "step"
6) Check contents of the structure "usb_data_buffer"
7) Evaluate the variable "device_data"

Programming \ Examples \ Libraries \ Data access and data storage \ Data storage

After enabling the program via the variable "start_reading_usb_data", connected USB mass storage devices are determined with the controller. A file device with the name "DEVICE1" is generated for the first USB flash drive. Then the content of the USB flash drive is read with the function blocks DirInfo() and DirRead(). The file device is enabled again at the end.

6.3 mapp Technology recipe management

With mapp Technology[4], we offer users an easy-to-use interface for implementing comprehensive functionality. Many complex operations, such as loading and saving recipe data, controlling a drive axis and recording process values, can be implemented quickly and easily using mapp Technology components.

Figure 38: mapp Technology logo

mapp Technology unites configuration and programming. Functionality is implemented in the application program using standard libraries. In addition, mapp provides configuration interfaces that allow the functionality of mapp components to be influenced independently of where they are used in the application software.

Application layer - mapp Technology

- Concept
- Getting started
- Components

Recipe management using mapp Technology

The MpRecipe component provides all of the functions necessary for simple yet high-speed recipe management. This includes reading and writing recipe files as well as a connection to the HMI application.

The MpRecipeRegPar function block can be used to register process variables for recipe management. The MpRecipeXml function block is then used to load recipe files from or save them to a file device. The MpRecipeUI function block provides a complete interface for visualization with Visual Components in order to expand functionality.

Communication between the individual components takes place via the MpLink, which is implemented by adding the standard configuration for the recipe component in the Configuration View.

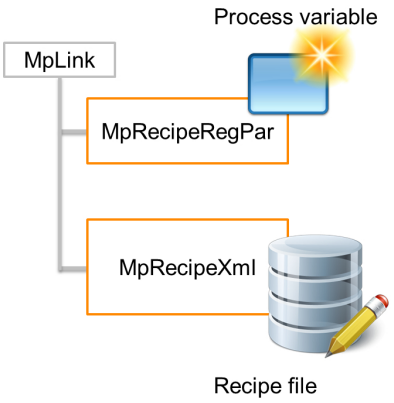


Figure 39: Registering a process variable for recipe management; saving and loading a recipe file

Diagnostics via WebXs

All mapp Technology components provide the possibility for web-based diagnostics. This WebXs (Web Access) shows the instances of the mapp function blocks, as well as all input values, output values and status information. WebXs (Web Access) is handled using System Diagnostics Manager.

4 mapp Technology stands for "Modular APPlication technology".

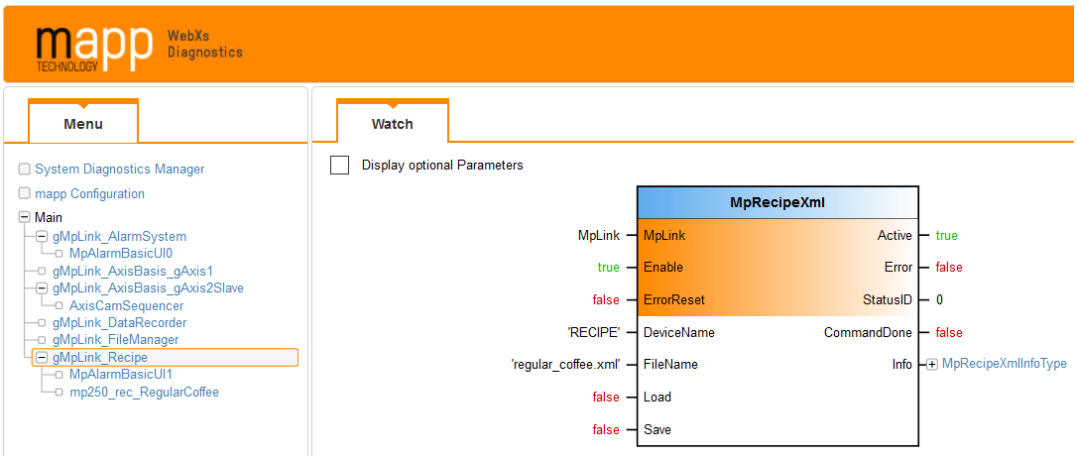


Figure 40: Diagnosis of the MpRecipeXml component using the automatically generated WebXs

Application layer - mapp Technology

- Concept
- Getting started \ Recipe management with visualization
- Components \ Infrastructure \ MpRecipe - Recipe management

Task: Save and load the recipe structure

The recipe component is used to save and download the recipe. It is necessary to specify an existing file device and the desired filename and extension. The same MpLink is used for the ensuing registration of the recipe parameters.

1) Configure the "RECIPE" file device for the "C:\Recipe" folder.
2) Call the recipe component (MpRecipeXml).
3) Operate the recipe component using commands.
   - Use "cmdSaveRecipe" to save the recipe.
   - Use "cmdLoadRecipe" to load the recipe.
4) Evaluate the status ID and the "CommandDone" output.

The recipe that's now saved doesn't contain any data yet. The required recipe parameters are registered in the next step.

Task: Register recipe structure as recipe parameters

To save recipe files, the recipe parameters on the recipe component must be registered.

The "RegularCoffee" variable is used as the data for the recipe file. This uses the data type "recipe_typ" which was generated in a preceding exercise. After the recipe parameters have been successfully registered, you can save the recipe.

1) Declare the "RegularCoffee" variable as type "recipe_typ" and use in the program code
2) Register the recipe data on the recipe system - MpRecipeRegPar()
3) Check the status ID.
4) Save the recipe and check the result

1) If the application program is created with the help of a state machine, make sure that all mapp function blocks are called at the end of the program.
2) The optional function block parameters can be used to extend the basic functionality if desired.
3) When registering local variables, the task name should be specified in the following syntax: "<Task name>:<Variable name>"

The Structured Text program for saving and loading the recipe could look like the following:

```
(*use MpRecipe to save and load recipe for "RegularCoffee"*)
MpRecipeXml_0.MpLink := ADR(gRecipeXml);
MpRecipeXml_0.Enable := TRUE;
MpRecipeXml_0.ErrorReset := cmdErrorReset;
MpRecipeXml_0.DeviceName := ADR('RECIPE');
MpRecipeXml_0.FileName := ADR('coffee.xml');
MpRecipeXml_0.Load := cmdLoadRecipe;
MpRecipeXml_0.Save := cmdSaveRecipe;

(*register "RegularCoffee"*)
MpRecipeRegPar_0.MpLink := ADR(gRecipeXml);
MpRecipeRegPar_0.Enable := MpRecipeXml_0.Active;
MpRecipeRegPar_0.ErrorReset := cmdErrorReset;
MpRecipeRegPar_0.PVName := ADR('read_data:RegularCoffee');

(*Call mapp function blocks*)
MpRecipeXml_0();
MpRecipeRegPar_0();
```

Exercise: Save recipe to a USB flash drive

The preceding tasks can be combined sensibly. The program for saving the recipe is added to the program for reading the USB flash drive. The program execution is also modified somewhat.

1) Add program code with MpRecipeXml() and MpRecipeRegPar() to the end of "LibAsUSB1_ST"
2) Change file device for MpRecipeXml() to "DEVICE1"
3) Replace content of step "DIRECTORY_READ" with treatment of "cmdSaveRecipe"
4) Transfer the program
5) Insert the USB flash drive into the controller

6) Execute by setting "start_reading_usb_data"; observe behavior of "step"

7) Check the recipe file on the USB flash drive

Instead of the function block DirRead(), the command "cmdSaveRecipe" is dealt with. The status outputs are evaluated by the function block MpRecipeXml() "CommandDone" and "Error" for automatic program execution. After successfully saving the recipe, the next step is switched to and "cmdSaveRecipe" is reset.

```
DIRECTORY_READ:
    cmdSaveRecipe := TRUE;
    IF MpRecipeXml_0.CommandDone = TRUE THEN
        cmdSaveRecipe := FALSE;
        step := UNLINK_DEVICE;     (*saving successful, unlink File Device*)
    END_IF

    IF MpRecipeXml_0.Error = TRUE THEN
        step := ERROR_CASE;       (*error occured*)
    END_IF
```

Exercise: Enable and verify the WebXs (Web Access)

The mapp components provide automatic WebXs (Web Access) for diagnostic purposes. WebXs is enabled by assigning the MpWebXs library in the software configuration. When the project has been transferred, the diagnostic and configuration contents of the respective mapp component are displayed in the web browser.

1) Import the MpWebXs library into the Logical View.

2) Assign MpWebXs to the software configuration.

3) Transfer the project and restart the controller.

4) Call the WebXs (Web Access) using System Diagnostics Manager.

5) Check the status ID of the recipe component in the WebXs.

6.4 Databases

Data objects and files can be used to easily store and archive data. For these to be directly integrated into existing IT infrastructure, accessing databases is often necessary. The "AsDb" library can be used to set up connections to SQL databases.

Programming \ Libraries \ Data access and data storage \ AsDb

7 Data transfer and communication

The topic of data transfer and communication is quite complex and comprises several different areas. Before approaching the implementation of a task, it's a good idea to familiarize yourself with the different aspects involved.

Questions to keep in mind with regard to data transfer and communication:

- Which data should be transferred?
- With what station am I communicating?
- Is it a PC or a controller?
- Which medium is being used?
- Which protocol is being used?
- Which data format is being used?
- Are there different platforms involved?
- Is there a standard library for the required protocol?
- Is there a fieldbus involved?
- How much data is to be transferred?
- What are we looking at with regard to transfer speed and response time?

This list of questions highlights the need for further clarification. Only then can implementation begin.

The Automation Help provides an overview of the different types of communication available.

It contains information about the means of transfer, protocols as well as the configurations and required libraries that can be used with them.

Communication

Programming \ Libraries \ Communication

Figure 41: Automation Help - Communication

7.1 General information about communication

If you take a closer look at the connections and how individual components are networked, you will realize that there are different topologies, access methods and transfer protocols. The topology indicates the type of connections used for communication nodes. Different topologies are supported according to the transfer medium. The transfer protocol transports the data.

The access method (Layer 1 + 2) indicates the coordination of network stations in a communication network. Each station on the network has a unique identifier (station or node number, IP or MAC address). The medium often determines the topology to be used, but not necessarily the access method or transfer protocol (Layer 3 + 4).

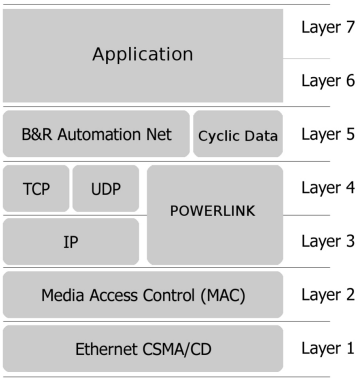


Figure 42: The OSI layer model[5] for TCP/IP or POWERLINK communication in Automation Runtime

5 http://en.wikipedia.org/wiki/OSI-model

Transfer media

The transfer medium describes the type of physical connection between communication nodes. This can include cables, signal levels and contacts (plugs). Different transfer distances are possible and certain topologies support based on the medium.

Frequently used transfer media:

- Serial (RS232 / RS485 / RS422)
- CAN
- Ethernet

Topologies

The topology is the type of connection between communication nodes. In the most basic form, the communication nodes are connected directly via the transfer medium. Connection types with greater complexity allow the communication nodes to be connected in a more flexible manner. The following table lists some common topologies and the supported transfer media. The subsequent figure illustrates the basic structure of the individual topologies.

Topology	Supported transfer media
Point-to-point	Serial, CAN, Ethernet
Bus	CAN, RS485, RS422
Line	Ethernet
Tree	Ethernet
Star	Ethernet
Ring	Ethernet

Table 39: Overview of topologies and respective transfer media

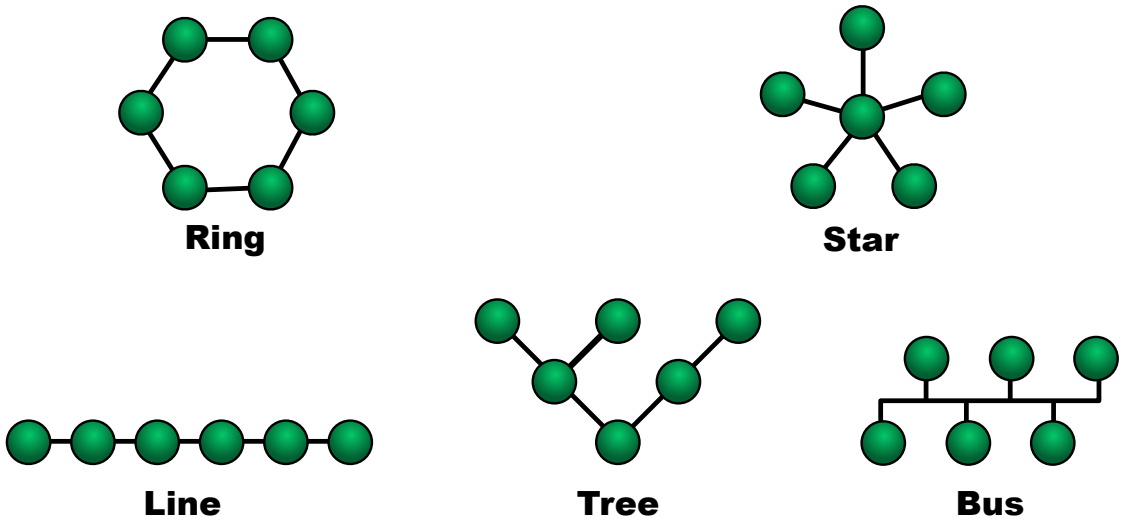


Figure 43: Illustration of commonly used topologies

Access methods

Only one station in a network can send data at a time. This is why an access method must be used which coordinates station communication on the communication medium.

Frequently used access methods:

- Master - Slave
  In a point-to-point connection, there is usually a master and a slave. The master issues commands to the slave, which responds to them or carries them out.
- Token Ring
  In token ring networks, one token circles through the ring. The token is passed on from one node to the next.
- CSMA CA / CD
  A network generally contains several stations. Depending on the access method, the stations on the network are coordinated differently. For example, there are access methods where collisions that occur are detected and resolved (CSMA / CD).[6] There are also access methods in which network collisions are prevented. (CSMA / CA)[7].
- Time slot method
  Ethernet POWERLINK, which is based on the Ethernet transfer medium, uses a time slot method. Therefore, there is no need to detect nor to prevent collisions. This method of transfer is real-time capable.

Fieldbus systems

Fieldbus systems are used when connecting I/O systems or other devices to controllers. The definition of a fieldbus system primarily consists of the definition of the transfer medium, the supported topologies, the access method and a transfer protocol. A fieldbus device can be inserted and configured in Automation Studio. The description of the device is usually provided in a generic format (.eds, .gsd, .xdd file) that can be imported directly into Automation Studio. You can read more about the integration of fieldbus systems in the Automation Studio help system.

Communication \ Fieldbus systems

Transfer protocols

The transfer protocol defines how data is transferred from A to B. The receiver must know how the transfer protocol is structured in order to correctly interpret the data received. Data must be transferred in the right format.

Elements of a transfer protocol include the following:

- Identifier of the sender
- Destination address of the receiver
- The amount of data
- Data
- Checksums

6 Carrier Sense Multiple Access / Collision Detection (CSMA / CD)
7 Carrier Sense Multiple Access / Collision Avoidance (CSMA / CA)

The transfer protocol TCP/IP[8] is often used for communication between controllers and PCs in a network. Collisions are detected and resolved using the CSMA/CD access method. Data is transferred quickly, however the time needed to resolve collisions compromises real-time capability. Ethernet POWERLINK is used to connected controllers, remote I/O modules, drives and integrated safety technology. The time slot method used by Ethernet POWERLINK[9] prevents collisions from occurring on the network. The result is data transfer which is both fast and real-time capable.

Communication

Communication \ POWERLINK

7.2 Communication libraries

Automation Studio includes standard libraries that handle communication. All of the function blocks have an enable input and a status output. The values of the status outputs are pre-defined as constants and described in the help documentation for the respective library.

Establishing communication requires first opening or initializing an interface. For connection transfers, such as client-server communication via TCP/IP, a connection to the other station(s) needs to be established. If communication is no longer needed, then the connection is terminated, the interfaces closed and the respective system resources freed up again.

Library examples are available for the majority of libraries and can be imported in the Logical View in Automation Studio. The respective documentation can be found in Automation Help.

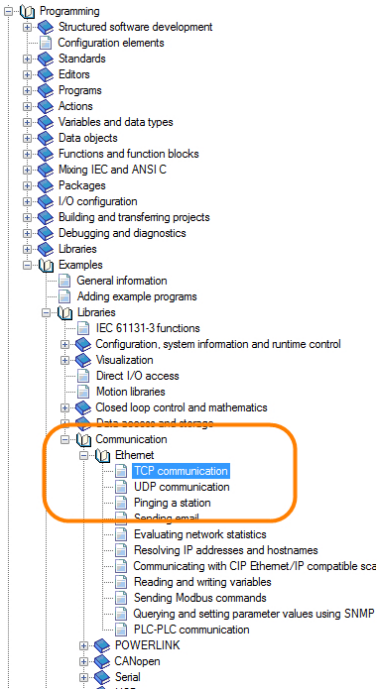


Figure 44: Documentation for the communication library examples in the Automation Studio help

Programming \ Libraries \ Communication

Programming \ Examples \ Libraries \ Communication

Diagnostics and service \ Error numbers \ AR system

Application possibilities

Using communication libraries allows many different application scenarios to be implemented. The AsTCP library, for example, makes it possible to set up a TCP/IP connection. 3rd-party devices such as controllers, IP cameras and PCs can also be connected to a B&R controller. The diagram offers a basic

[8] Transmission Control Protocol (TCP) based on the Internet Protocol (IP)

[9] http://www.ethernet-powerlink.org/

representation of which function blocks are needed to establish the connection and for communication. A socket is opened on the server station to set up a server. The client also opens a socket and connects to the server. Communication between the two stations can be achieved using the send and receive functions. This function is already implemented in the "LibAsTCP1_ST" library example, which can be imported in the Logical View in Automation Studio.

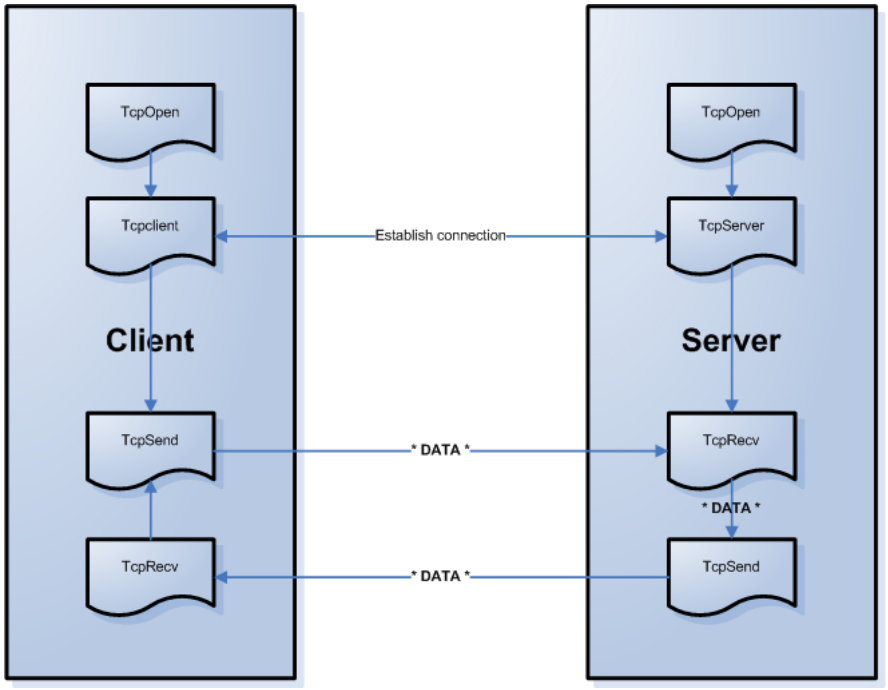


Figure 45: Diagram showing Client-Server communication with the AsTCP library and how a connection is established

Task: Communication over TCP/IP

1) Define the data.

   Use data from your controller project. Make sure that the structure and the data format are the same as the data from the station you are communicating with.

2) Import an example project.

   Import the example project "LibAsTCP1_ST" using the Logical View.

3) Establish the connection to your communication partner.

   Choose which station the client and server programs should be loaded to Alternatively, the client and server applications can be loaded to the same controller or in the simulation environment.

4) Testing communication

   Test whether the data is being correctly transferred from the client to the server and vice versa. You can do this using the variable monitor or the variable oscilloscope.

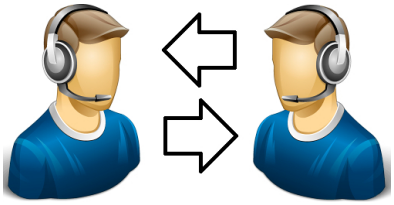


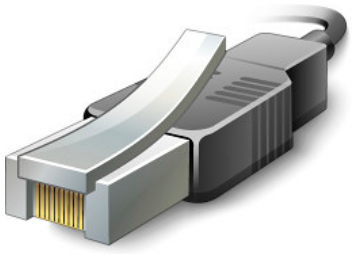
Figure 46: TCP/IP communication

5) Test scenarios in which communication fails.

   Break the connection to each controller in turn and test whether it can be reestablished by the communication program on its own.

8 Connectivity and access & security

The B&R solution is completely open for seamless integration in existing networks and allows direct integration in devices from other manufacturers. B&R components can be implemented as either fieldbus masters or slaves.



An OPC UA server and client enable the publication and exchange of process data. The user roles system helps with the limitation of read and write access. An encrypted connection is enabled using Transport Layer Security (TLS).

Support of devices from other manufacturers

Automation Studio allows fieldbuses and also devices from other manufacturers to be directly integrated. Master and slave devices are added to Automation Studio – like all other B&R components – in the Physical View or in the System Designer.

? Communication \ Fieldbus systems

Access & security - User management system

Automation Studio supports configuration of a user system, a role system and a certificate management system as well as management of SSL/TLS configurations. These configurations are managed in the "Access & Security" package in the Configuration View.

? Access & Security \

- User role system
- TLS / SSL

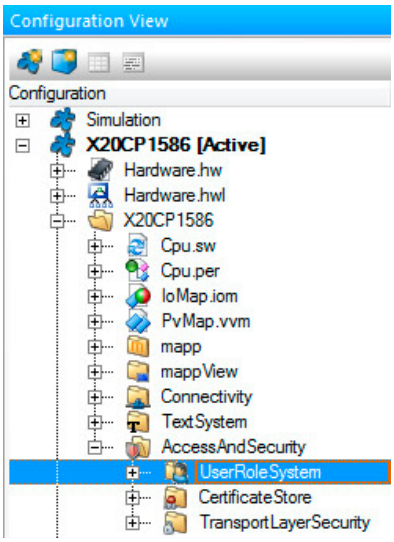


Figure 47: User management system in the Configuration View

OPC UA server and client

Automation Studio supports the configuration of an OPC UA server. Read/write access for the individual OPC UA codes is managed using the user role system. OPC UA client functions are offered via a library.

? Communication \ OPC UA

Libraries / Communication / AsOpcUac

Automation Software \ Sample programs \ OPC UA project

8.1 Support of devices from other manufacturers

Automation Studio allows fieldbuses and also devices from other manufacturers to be directly integrated. Master and slave devices are added to Automation Studio – like all other B&R components – in the Physical View or in the System Designer.

First a device description file is imported, then the device can be selected from the Hardware Catalog and added. These are the following configuration variants:

- Support for netx - FDT/DTM technology
- Configuration with device description file

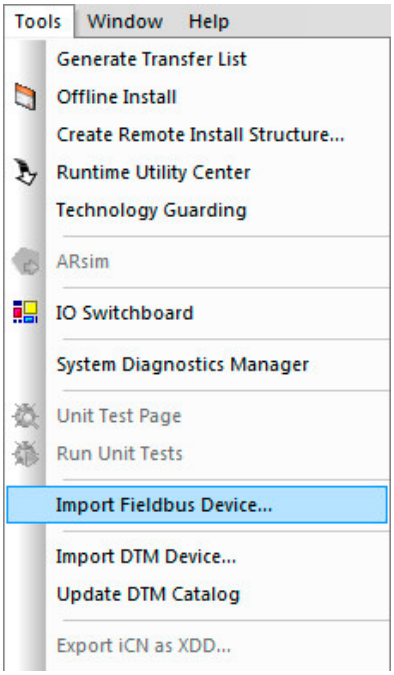


Figure 48: Import fieldbus device into Automation Studio

FDT/DTM technology

Field Device Technology (FDT) is a standardized interface between field devices and host systems. A Device Type Manager (DTM) is a vendor-specific user interface used to configure and troubleshoot a specific fieldbus module, which is called via the context menu in the Physical View.

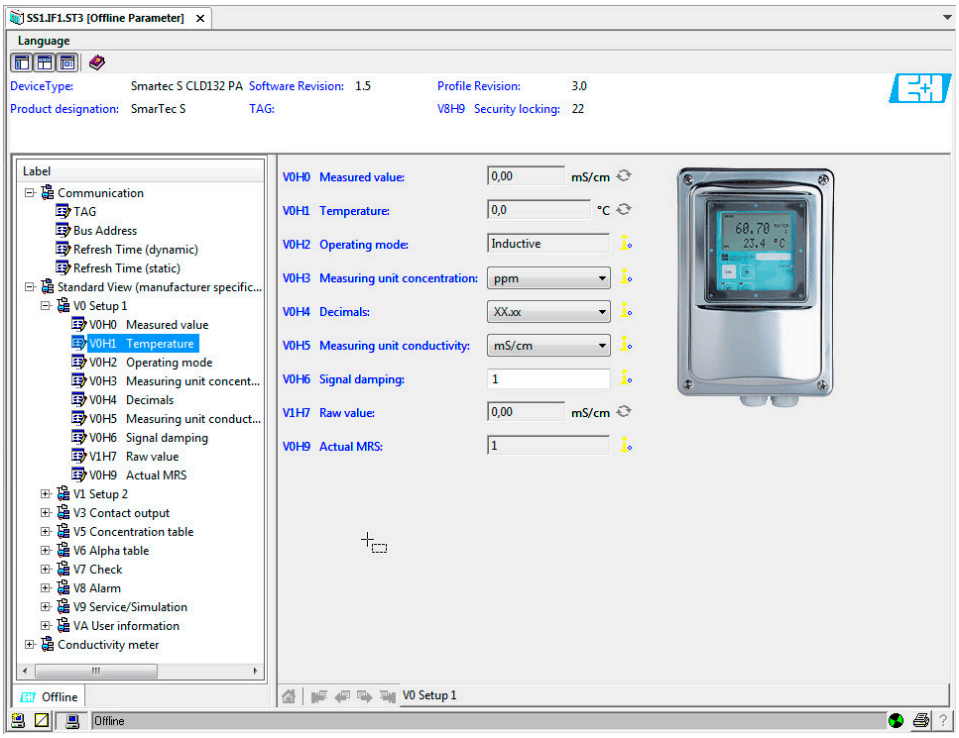


Figure 49: Device Type Manager (DTM) for vendor-specific configuration and diagnostics

Configuration with device description file

A fieldbus master (e.g. CANopen master) is an interface that can be inserted in Automation Studio in the Physical View. A fieldbus slave can be inserted directly on the master interface and then configured once a device description file has been imported. Configuration occurs as usual via the I/O configuration which is generated generically from the device description file.

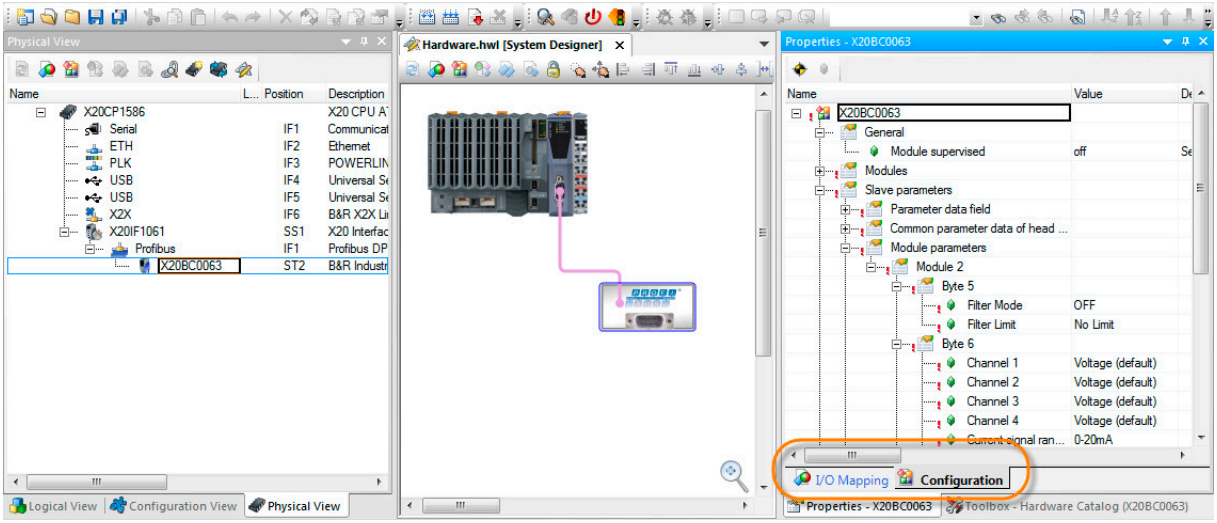


Figure 50: Configuration of a fieldbus participant

I/O mapping

Independent of the selected configuration variant, all fieldbus participants have an I/O mapping. The I/O data is available on the assigned process variables after the fieldbus participants have been configured by Automation Runtime.

Programming and diagnostics

When configuring using the device description file, diagnostics occurs via data points in the I/O mapping. When configuring via the FDT/DTM interface, diagnostics is also available via the Device Type Manager. Additionally, libraries can be used for all supported fieldbuses. Details about startup, status and errors is displayed in the "Fieldbus" Logger module.

Communication \ Fieldbus systems

- NetX - FDT/DTM
- Insertion with a device description file

Programming \ Libraries \ Communication

Diagnostics and service \ Diagnostic tools \ Logger window

8.2 Access & security - User management system

Automation Studio supports configuration of a user system, a role system and a certificate management system as well as management of SSL configurations. These configurations are managed in the "Access & Security" package in the Configuration View.

Adding in the "Access & Security" package

The "Access & Security" package is moved via drag-and-drop from the Toolbox into the Configuration View. The underlying configuration elements are set up automatically. The "UserRoleSystem" package is added, among others. This already encompasses the configuration of the "Administrators" and "Everyone" roles. Each user with one of the two roles was also set up.

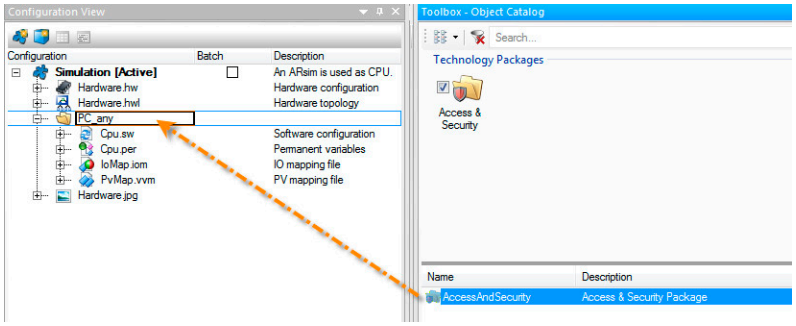


Figure 51: Adding in the "Access & Security" package

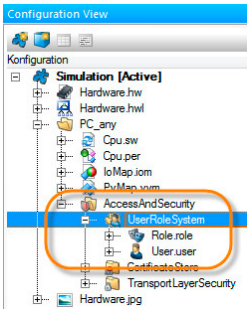


Figure 52: "UserRoleSystem" configuration package was automatically added

Access & Security \ User roles system

- Automation Studio configuration
- Configuring users
- Configuring roles

Programming \ Libraries \ Configuration, system \ ArUser

Application Layer - mapp Technology \ Infrastructure \ MpUserX - User management using the user roles system

Exercise: Enable user management system in the project

User management must be enabled in the project. In addition, the "Access & Security" package is added in the Configuration View.

1) Enable the Configuration View
2) Add the "Access & Security" package in the Configuration View

8.3 OPC UA server and client

OPC UA is a data model based on a communication protocol for client-server communication between two devices (e.g. controllers, industrial PCs, process control systems) from different manufacturers in accordance with the client-server principle. OPC UA is a software interface, not a bus system.

Automation Studio supports the configuration of an OPC UA server. Read/write access for the individual OPC UA nodes is managed using the user role system. OPC UA client functions are offered via a library.

Enable and configure OPC UA server

The OPC UA system must be configured in order to grant OPC UA clients access to data. First the OPC UA system in the CPU configuration is enabled. For basic configuration, it's sufficient to enable the OPC UA system. The remaining parameters stay unchanged. Likewise, certificates for the protection of client-server communication can be enabled here.

Finally, the global and local process variables, which should be visible as OPC UA tags on the client, are enabled.

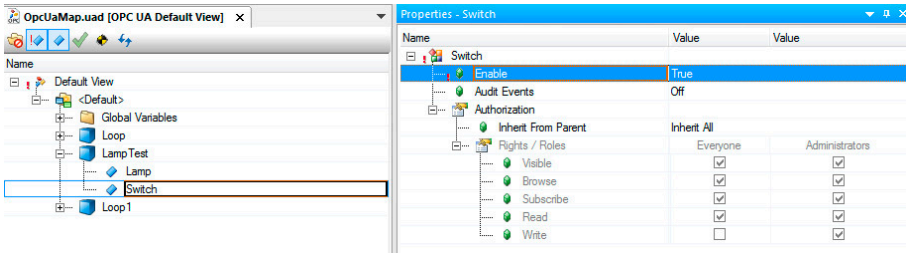


Figure 53: OPC UA default view: Enable the "Switch" variable - Access rights are inherited

If no roles are assigned in the default view, then there are no OPC UA nodes visible on the OPC UA client.

The following help articles aid with OPC UA server configuration and the settings in the OPC UA Default View.

Communication \ OPC UA \

- Enable the server
- Default view configuration
  - Add in default view configuration
  - Default view editor
  - Default view editor \ Properties \ Authorization

Access & Security

OPC UA server diagnostics

All events that occur on the OPC UA server are logged in the "Connectivity" Logger file. For example, if write access is granted on the OPC UA client, but with missing write access rights to an OPC UA tag, then this is logged in the Logger. The image shows that write access to the "Switch" variable was attempted. Write access was rejected by the server due to missing write access.

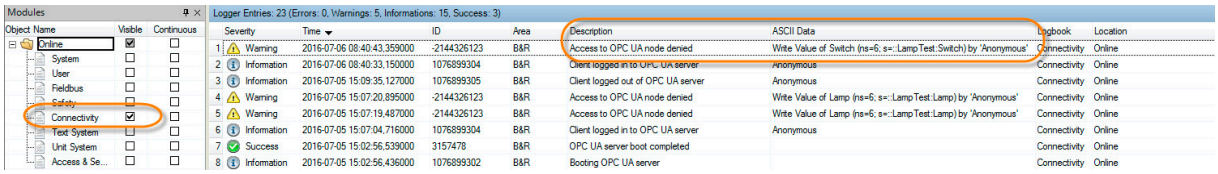


Figure 54: OPC UA server diagnostics; Write access to "Switch" was rejected by the server

Diagnostics and service \ Diagnostic tools \ Logger window

OPC UA client function blocks

The AsOpcUac library can be used to exchange the process data on a B&R controller with OPC UA servers on systems of different platforms. The library's function blocks for OPC UA client functionality were developed in cooperation between OPC Foundation and PLCopen working groups. A sample project for using the function examples is installed with Automation Studio.

Programming / Libraries / Communication / AsOpcUac

Automation Software \ Example programs \ OPC UA project

Exercise: Variables from "LampTest" program with OPC UA client

The goal of this exercise is to read the "Lamp" variable from the "LampTest" program using OPC UA client software. To this end, the OPC UA default view is added to the Configuration View, roles for read/write access are assigned and the required process variables are enabled.

1) Add OPC UA default view to Configuration View
2) Add "Everyone" and "Administrators" roles
3) Enable the "Lamp" variable
4) Transfer the configuration
5) Access to OPC UA node "Lamp" with OPC UA client software

   see: "Establishing a connection with Unified Automation UaExpert client" on page 68
6) Grant write access to "Lamp": Check the "Connectivity" Logger file

Exercise: Switch on "Lamp" output via OPC UA client

The lamp is currently switched on via the "Switch" variable that's connected with a digital input. In order to also control the lamp via the OPC UA client, an OR operator with the variable "vCtrlSwitch" is added in the Ladder Diagram program.

1) Expand "LampTest" program by OR connective with variable "vCtrlSwitch"

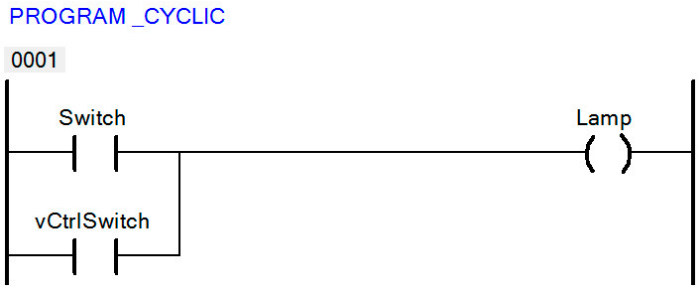


Figure 55: Expansion of "LampTest" program with OR connective of variable "vCtrlSwitch"

2) Enable variable "vCtrlSwitch" in the OPC UA default view and check write access for the "Administrators" role

3) Display OPC UA node "vCtrlSwitch" using OPC UA client software and set the value to "TRUE".

8.4 Establishing a connection with Unified Automation UaExpert client

The following steps should be performed if the UaExpert client software from Unified Automation is used for establishing an OPC UA client connection.

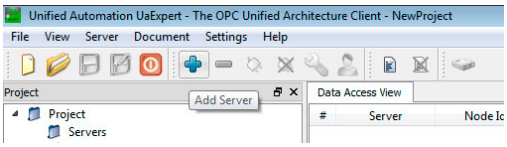


Figure 56: UaExpert: Add a new server via the toolbar

1) Add server
2) Enter server URL
3) Select signature and encryption
4) Accept server certificate
5) Move OPC UA nodes to Data Access View

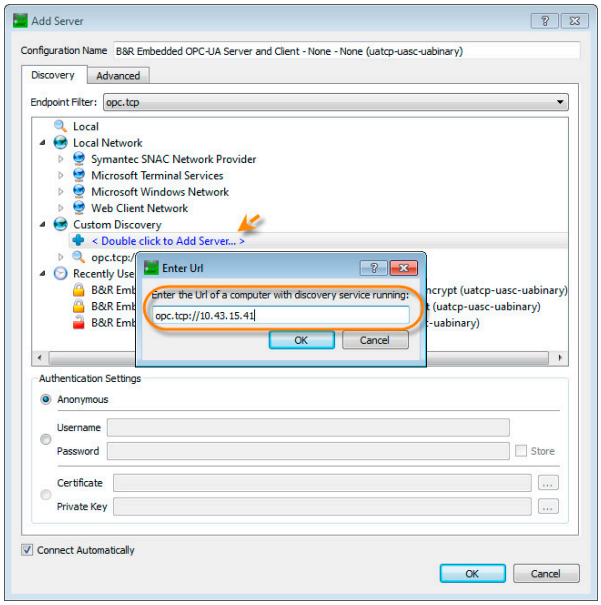


Figure 57: UaExpert: Enter server URL

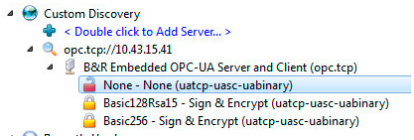


Figure 58: UaExpert: Select signature and encryption

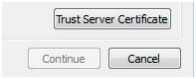
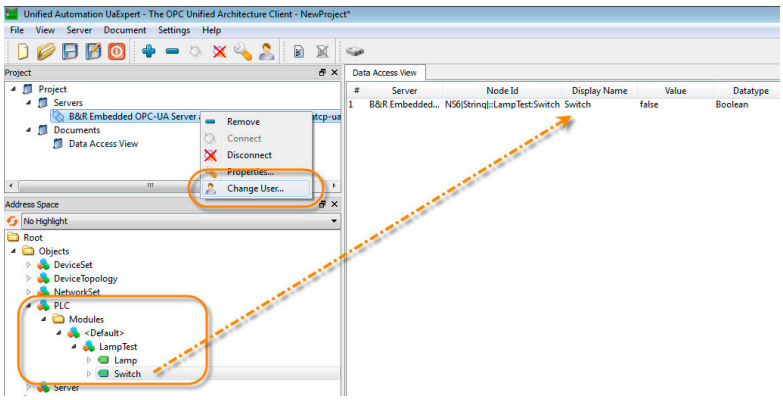


Figure 59: UaExpert: Accept server certificate



The required OPC UA nodes are moved to the Data Access View using drag-and-drop.

The user can be changed in the shortcut menu for the server. User-name and password depend on the configured roles in the OPC UA Default View in the Automation Studio project.

Figure 60: UaExpert: Move UA nodes to Data Access View; Change user

Detailed instructions about UaExpert client use can be looked up in the UaExpert help system in the "First Steps" chapter.

9 Summary

Anyone who is responsible for designing a control application is constantly confronted with data. Data is represented by variables and constants, regardless of the programming language. A variety of basic IEC data types, arrays and structures are available for this purpose. Automation Studio features a user-friendly table editor for easily initializing constants, variables and arrays. Function blocks and functions are provided for managing memory blocks in the application. The basic element of data processing is the storage format of the data on the respective platform. Functions are available for determining and using memory addresses and memory sizes in the program.

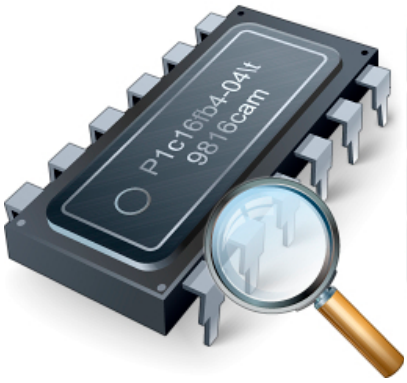


Figure 61: Memory and variables

Figure 62: Data processing and storage

Figure 63: Communication and data exchange

Automation Studio provides complete standard libraries and mapp Technology components for saving process data. Communication with other stations is made possible by the support of fieldbus systems and complete network integration. Standard libraries and an OPC UA server are provided for this purpose. Thanks to an extensive help system and library examples, integrating system functions in your applications is made much easier.

Seminars and training modules

At the Automation Academy, you'll develop the skills you need in no time!
Our seminars make it possible for you to improve your knowledge in the field of automation engineering.

Automation Studio seminars and training modules

Programming and configuration	Diagnostics and service
SEM210 – Basics SEM246 – IEC 61131-3 programming language ST* SEM250 – Memory management and data storage SEM410 – Integrated motion control* SEM441 – Motion control: Electronic gears and cams** SEM480 – Hydraulics** SEM1110 – Axis groups and path-controlled movements** SEM510 – Integrated safety technology* SEM540 – Safe motion control*** SEM610 – Integrated visualization*	SEM920 – Diagnostics and service for end users SEM920 – Diagnostics and service with Automation Studio SEM950 – POWERLINK configuration and diagnostics* If you don't happen to find a seminar on our website that suits your needs, keep in mind that we also offer customized seminars that we can set up in coordination with your sales representatives: SEM099 – Individual training day Please visit our website for more information**** .****: **www.br-automation.com/academy**

Overview of training modules	
TM210 – Working with Automation Studio TM213 – Automation Runtime TM223 – Automation Studio diagnostics TM230 – Structured software development TM240 – Ladder Diagram (LD) TM241 – Function Block Diagram (FBD) TM242 – Sequential Function Chart (SFC) TM246 – Structured Text (ST) TM250 – Memory management and data storage TM400 – Introduction to motion control TM410 – Working with integrated motion control TM440 – Motion control: Basic functions TM441 – Motion control: Electronic gears and cams TM1110 – Integrated motion control (axis groups) TM1111 – Integrated motion control (path controlled movements) TM450 – Motion control concept and configuration TM460 – Initial commissioning of motors TM500 – Introduction to Integrated Safety Technology TM510 – Working with SafeDESIGNER TM540 – Integrated safe motion control	TM600 – Introduction to visualization TM610 – Working with integrated visualization TM611 – Working with mapp View TM630 – Visualization programming guide TM640 – Alarm system, trends and diagnostics TM670 – Advanced Visual Components TM920 – Diagnostics and service TM923 – Diagnostics and service with Automation Studio TM950 – POWERLINK configuration and diagnostics TM280 – Condition Monitoring for vibration measurement TM480 – The Basics of Hydraulics TM481 – Valve-based hydraulic drives TM482 – Hydraulic servo pump drives TM490 – Printing machine technology In addition to the printed version, our training modules are also available on our website for download as electronic documents (login required): Visit our website for more information: **www.br-automation.com/academy**

Process control seminars and training modules

Process control standard seminars	Process control training modules
SEM841 – Process control training: Basics 1 SEM842 – Process control training: Basics 2 SEM890 – Advanced process control solutions	TM800 – APROL system concept TM810 – APROL setup, configuration and recovery TM811 – APROL runtime system TM812 – APROL operator management TM813 – APROL web portal TM820 – APROL solutions TM830 – APROL project engineering TM835 – APROL ST-SFC configuration 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 TM880 – APROL reporting TM890 – The basics of LINUX

* SEM210 - Basics is a prerequisite for this seminar.
** SEM410 - Integrated motion control is a prerequisite for this seminar.
*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.
**** Our seminars are listed in the Academy\Seminars area of the website.
**** Seminar titles may vary by country. Not all seminars are available in every country.