Memory Management and Data Storage

TM250

Requirements

Training modules:	TM213 - Automation Runtime TM246 - Structured Text (ST)
Software	Automation Runtime 3.08 Automation Studio 3.0.90
Hardware	None

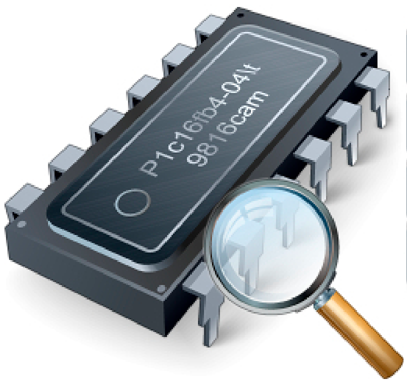
TABLE OF CONTENTS

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.



Memory and variables

Data processing and storage

Communication and data exchange

As a whole, the information presented here is meant to help determine which tools and procedures are available for processing and defining data.

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.

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.

1.1 Objectives

You will be learning about the following:

- ... Initializing and using variables
- ... Using enumerated and user-defined data types
- ... The correct usage of B&R standard libraries
- ... How to create a user library
- ... The basics of data preparation and processing
- ... Basic information about data formats
- ... An overview of the different ways that data can be stored
- ... Information about different methods of communication
- ... Using B&R library samples

2 VARIABLES, CONSTANTS AND DATA TYPES

Fixed memory addresses have long been a thing of the past in programming; these days, programming takes place using uniquely named symbolic elements. These elements are called 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, errors and user-defined data types.

2.1 The basics

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.

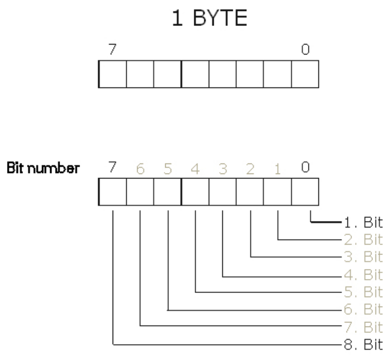
If you can understand this system, it makes programming a system and localizing errors that may occur much easier.

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.

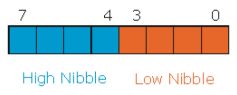
The bits within a byte are numbered from right to left, from **Bit 0** to the most significant value **Bit 7**. Bit 2 (which is actually the 3rd bit) therefore has a decimal value of 4.



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.



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	$2^{BitNumber}$
00000001	0	1	2^0
00000010	1	2	2^1
00000100	2	4	2^2
00001000	3	8	2^3

Table: 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.

Bit pattern	Bit number	Decimal value	$2^{\text{BitNumber}}$
00010000	4	16	2^4
00100000	5	32	2^5
01000000	6	64	2^6
10000000	7	128	2^7

Table: The values of individual bits within a byte

Two binary numbers will be added together in this example.

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

Table: Addition of binary numbers with carry over

The following applies:

0 + 0 = 0

0 + 1 = 1

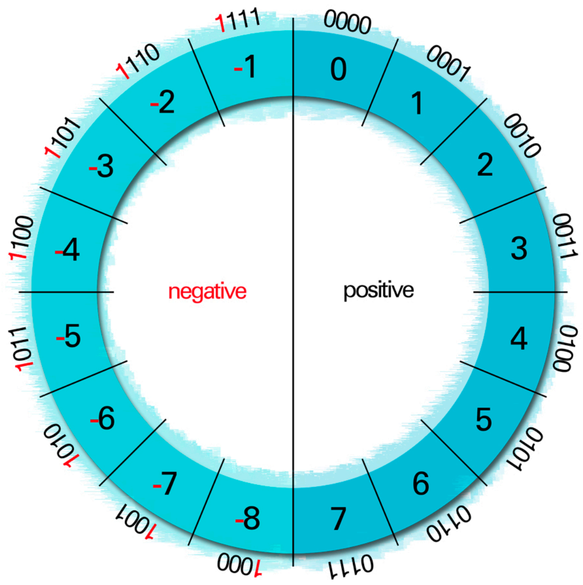
1 + 1 = 0 + carry over to the next bit

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: 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.



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: 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.

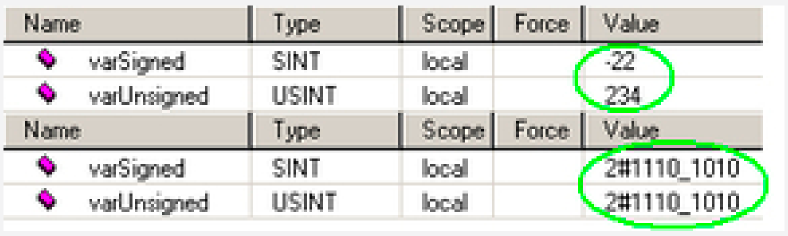
Declaration	`VAR`
	`    varUnsigned :  USINT := 0;`
	`    varSigned :  SINT := 0;`
	`END_VAR`

Program code	`varSigned := -22; (*Bit pattern 1110 1010 *)`
	`varUnsigned := varSigned;`

Table: 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).



Name	Type	Scope	Force	Value
varSigned	SINT	local		-22
varUnsigned	USINT	local		234

Name	Type	Scope	Force	Value
varSigned	SINT	local		2#1110_1010
varUnsigned	USINT	local		2#1110_1010

Different data types display the same data differently.

Programming \ Variables and data types \ Variables \ Data types \ Basic data types \ INT

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

Table: Converting binary numbers to hexadecimal

1101	Decimal 13	Hex D
1110	Decimal 14	Hex E
1111	Decimal 15	Hex F

Table: Converting binary numbers to hexadecimal

Nibbles and hexadecimal

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

Table: 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: 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 variable monitor 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 include things like limit values or range limits.

Any of the available basic data types can be used in either case (see Chapter 2.2).

Assigning a value to a constant

1) Declare the constant.

   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.

? Programming \ Variables and constants \ Variables \ Constants

Constants can also be used to initialize arrays (see Chapter 2.3.1). In order to do this, however, the project setting "Allow extension of IEC standards" needs to be enabled.

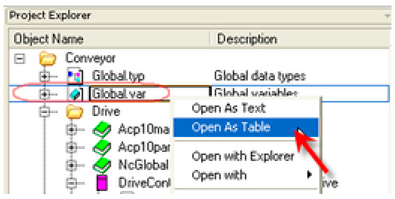
? Project Management \ The Workspace \ General project settings \ Settings for IEC compliance

2.1.3 Declaring variables, constants and data types

This training manual will not go into much detail about the Automation Studio user interface.

At this point, we would like to show you how to open the variable and data type declaration windows.

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



Opening the declaration window as a text or table editor

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 composite 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 \ SmartEdit

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 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 \ Scope of declarations

2.2 Basic data types

Primitive data types are also called basic data types. These data types form the basis for all other composite 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 series	Signed integers	Unsigned integers	Floating point	String	Time, date
BOOL	SINT	USINT	REAL	STRING	TIME
WORD	INT	UINT	LREAL	WSTRING	DATE_AND_TIME (DT)
DWORD	DINT	UDINT			DATE
					TIME_OF_DAY (TOD)

Table: Overview of IEC data types

A complete list of basic data types, their areas of use and range of values can be found in the Automation Studio online help documentation.

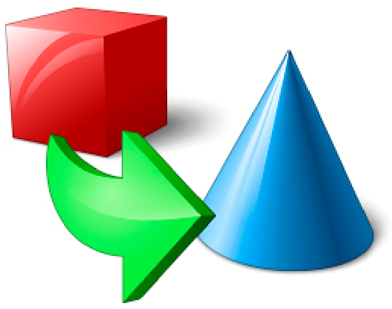
Programming \ Variables and data types \ Variables \ Data types

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.

2.2.1 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.


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: Implicit 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 a value of 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: 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: 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 derivation. User-defined data types consist of elements from the basic data types.

These derived data types include the following:

- Arrays and multi-dimensional arrays
- Structures
- Direct derivation and subranges
- Enumerations

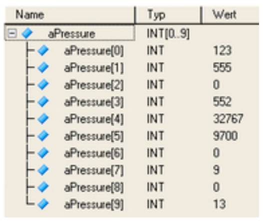
Programming \ Variables and data types \ Data types \ Derived data types

2.3.1 Arrays

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

The index used to access values in the array 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

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 the value 123 to index 0*)`
	`aPressure[0] := 123;`

Table: 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: 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]". In both of these cases, an array with a starting index of 0 and a maximum index of 9 will be created.

Creating the "aPressure" array

1) Create a new program called "arrays".
2) Open the variable declaration window.
3) Declare the "aPressure" array.

   The array should contain 10 elements. The smallest array index starts at 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 and constants 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.

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: 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

Calculating the sum and average value

The average value should be calculated from the contents of the "Pressure" 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) Calculate the average value.
   The data type of the average value must be the same as the data type of the array (INT).

Multi-dimensional 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`	

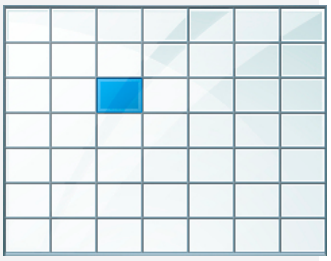
Program code	`Array2Dim[3,3] := 11;`	 Accessing the value in Column 3, Row 3

Table: 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 deravation and subranges

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

It is possible to derive these derived data types directly from the basic data types. Doing so creates a new data type with a new name that has the same properties as the basic 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 valid value range can also be specified for direct composite 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: Declaring a variable and data type with a subrange

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

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

Declaring a direct derived type with a subrange

A new data type "pressure_typ" should be derived. 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) Declare a direct composite 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.

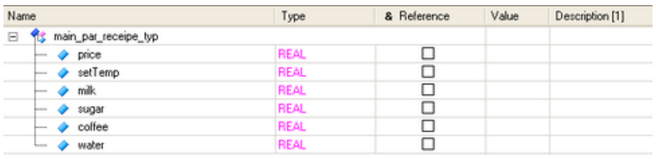
4) Test your results.

   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 Structures

A structure – also known as user-defined data type – is the grouping together of a collection of elements of basic data types or structures that are addressed using a shared name. Each of the individual elements has its own name.

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, depending on the recipe, in different amounts.



Name	Type	& Reference	Value	Description [1]
main_par_receipe_typ				
price	REAL	☐		
setTemp	REAL	☐		
milk	REAL	☐		
sugar	REAL	☐		
coffee	REAL	☐		
water	REAL	☐		

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: Declaring a structure, usage in the program code

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

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

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) Declare the structure.

   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" using 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 basic 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;
```

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.

2.3.4 Enumerated data types

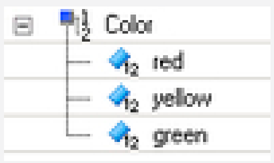
Enumeration is basically another way of referring to a list of values. It is therefore possible to use enumerated data types with variables. Instead of the value of the listed element, the variable contains the corresponding text.

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
```



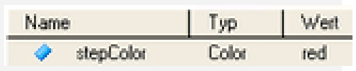
Enumerated type declaration

Variable declaration

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

Program code

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



Name	Typ	Wert
stepColor	Color	red

Enumerated type displayed in the variable monitor

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

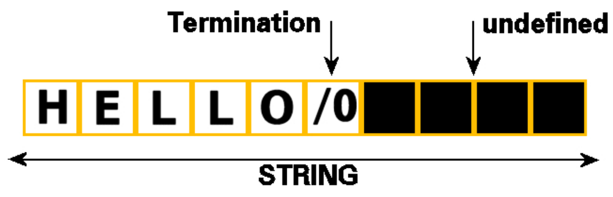
Programming \ Variables and data types \ Data types \ Derived data types \ Enumeration

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.



String of 10 characters, zero-termination after "Hello"

The table below shows how the 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] := 'Description';`
	`END_VAR`

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

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

If a string is assigned to the string variable that is too long (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 \ Basic data types \ STRING, WSTRING

Programming \ Standards \ Literals in IEC languages

Programming \ Standards \ ASCII tables

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.

3) Check the results in the variable monitor.

2.4.1 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

Most string operations require the use of library functions.

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 stringa = stringb THEN`
	`    equal :=  TRUE;`
	`ELSE`
	`    equal :=  FALSE;`
	`END_IF`

Table: 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

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: Converting from String to Real

Conversion in the reverse direction looks like this:

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

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

Table: Converting from 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: Determining the position of a string within another string

The result is the value 9 since the string being looked for begins at the 9th position in the string being searched.

Programming \ Libraries \ IEC 61131-3 functions - STANDARD \ STRING handling functions

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 and memory size

Addresses

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

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. see Chapter 4

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

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

Table: 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.

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

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 \ Address and length functions \ ADR

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

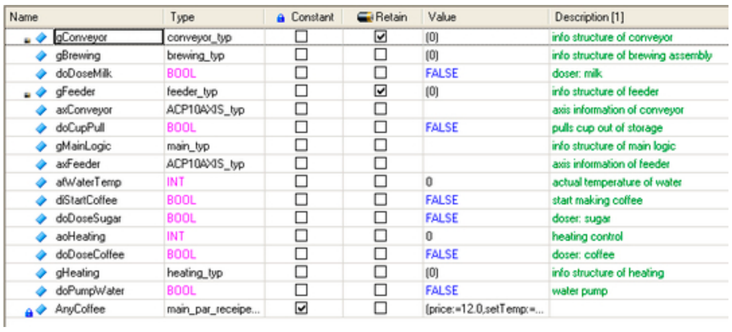
3 MEMORY AND MEMORY MANAGEMENT

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

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
afWaterTemp	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:=...	

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 in the controller's battery-buffered SRAM 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 input support

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

Copying

Depending on the programming language and environment being used, there are several different methods available for copying data. In Structured Text, 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: 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 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: Copying limited memory areas with "min()"

Initializing

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: Initializing a memory area with "brsmemset()"

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

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.

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.

4 WORKING WITH LIBRARIES

Libraries allow software to be packaged in compact units and reused whenever necessary. Libraries therefore act as containers for functions, 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.

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 \ Logical view \ Wizards in the logical view \ Inserting libraries

Function

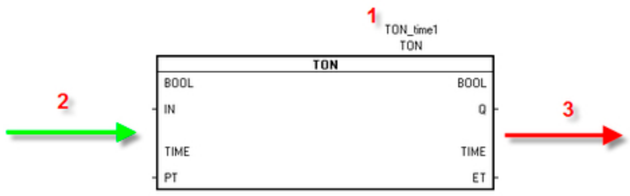
A function consists of the actual function call, the parameters being transferred and a value that is returned. When a function is called, the parameters are passed and a return value is returned immediately. Different parameters can be transferred the next time the function is invoked.

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

Table: 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 to have a function block perform a task by calling it several times.



Instance, inputs and outputs of 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.

Table: Image description

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: 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.

When a function or function block is inserted, the "F1" key can be used to open up the Automation Studio help documentation. It contains information about the parameters as well as the inputs and outputs for that particular function or function block.

Generating a clock signal

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

Programming \ Functions and function blocks \ Functions

Programming \ Functions and function blocks \ Function blocks

Calling function blocks: The enable input and status output

Function blocks included in B&R standard libraries all 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. Information about status values can be found in the Automation Studio online help documentation.

Constant	Error number	Meaning
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: 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.

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 := 1;`
	`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 handler here ... *)`
	`    END_IF`
	`END_IF`

Table: 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.

Calling a function block and evaluating the status

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.

2) Evaluate the status values.

   Evaluate the status outputs correctly. Set a variable to 1 if the status = ERR_FUB_BUSY. Set it 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 the Automation Studio online help documentation.

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

? Diagnostics and service \ Error numbers \ Libraries

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

Components of a library

? Programming \ Libraries

Project management \ Logical view \ Libraries

4.2 Creating user libraries

New user library

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.

The following things need to be considered:

- 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

The wizard in the logical view can be used to insert a new library. Libraries can be stored in the "Libraries" package or any other package in the logical view.

The following settings are possible:

- Name and description of the library
- Programming language
- Content such as constant declarations (.var)
- Data type declaration file (.typ)
- Function and function block declaration file (.fun)

? Project Management \ Logical view \ Wizards in the logical view

Inserting a function block

After it has been created, the library can be selected in the logical view. The wizard can be used to add a function or function block to a library.

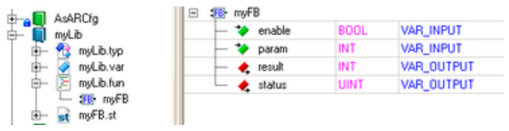
The following settings are possible:

- 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. Parameters can be handled just like normal variables.



myFB

enable	BOOL	VAR_INPUT
param	INT	VAR_INPUT
result	INT	VAR_OUTPUT
status	UINT	VAR_OUTPUT

Parts of the new library

If a function block outputs a status value, we recommend that constants be defined for it. These constants can be declared in the library's .var file. There is a defined user area provided for user constants.

Diagnostics and service \ Error numbers \ AR system \ 50000 - 59999 user area

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.

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. An enable input and status output must be included in the function block.

Name	Properties / Value	Data type
	Parameter (.fun)	

Table: Interface and constants for "adderx"

Name	Properties / Value	Data type
`enable`	`VAR_INPUT`	`BOOL`
`weight1`	`VAR_INPUT`	`INT`
`weight2`	`VAR_INPUT`	`INT`
`weight3`	`VAR_INPUT`	`INT`
`status`	`VAR_OUTPUT`	`UINT`
`sum`	`VAR_OUTPUT`	`DINT`
`average`	`VAR_OUTPUT`	`INT`
`Maximum`	`VAR_OUTPUT`	`INT`

Table: Interface and constants for "adderx"

Project Management \ Logical view \ Wizards in the logical view \ Inserting libraries

Project Management \ Logical view \ Inserting functions and function blocks into libraries

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

Project management \ Logical view \ Exporting a user library

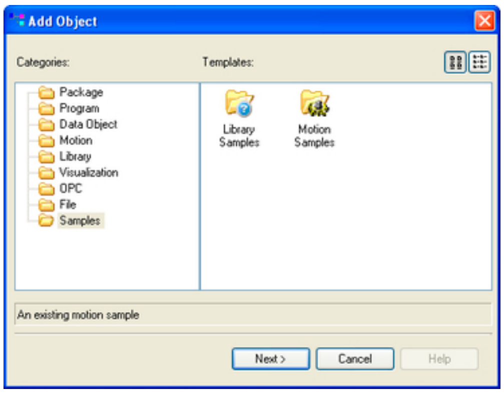
Project management \ Logical view \ Help for user libraries

4.3 B&R standard libraries samples

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 samples. 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 wizard in the logical view.

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



Wizard - Adding a sample

The samples were also designed so that they can be tested without any controller hardware actually being present. This is done with ARSim.

The structure of the individual programs is the same. This makes it easy to orient yourself to the different areas. A description and overview of the samples can be found in the Automation Studio online help documentation.

Programming \ Examples \ Adding samples

Programming \ Examples \ Libraries

Inserting a sample

Insert the following sample into your project:

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

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 stored or transferred.

5.1 Alignment

Different process architectures follow different rules when it comes to operations involving data and data management. 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.

Creating a user-defined data type

1) Modify the user-defined data types.

   Modify the data types of the elements of "recipe_typ". These elements are listed in the table.

2) Determine the size of the data.

   Declare a variable "SimpleCoffee" that uses the user data type "recipe_typ". Use the sizeof() function to determine the data length 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: Elements of the data type "recipe_typ"

In the previous task, the data length determined did not correspond to expectations. The additional data is in the form of stuff bytes. These are added automatically 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.

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.

Real-time operating system \ Target systems \ SG4 \ Runtime behavior \ Alignment

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.

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: 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 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 rather quickly in a single pass. 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));
  2: (*--- save data into file*)
    SaveData( ADR(aSendData));
END_CASE
```

Table: Preparing data before saving it to a file

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

Immediately before the save function is called, the modified data is copied from the application to a byte field. It remains consistent throughout the entire save procedure.

Programming \ Libraries \ Communication \ AsTCP \ General information

Programming \ Libraries \ Communication \ AsSnmp \ General information

Programming \ Libraries \ Data access and data storage \ FileIO

6 STORING AND MANAGING DATA

Data can be stored on the controller in many different ways. It can be managed as a memory block in DRAM, diverted as a B&R data object with checksum monitoring or stored locally in a file on a file system or network.

6.1 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 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 \ SYS_Lib \ Functions and function blocks \ Memory management

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

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

6.2 Data objects

B&R data objects

To configure different parts of the software, it is sometimes necessary to have parameter data available in a parameter file. These files can be read by the application and, if necessary, re-saved.

B&R data objects can be used in these cases. It is possible to save the files as well as write data to them directly in Automation Studio. At runtime, the checksums of these data objects are monitored.

This checksum monitoring makes it possible to detect corrupted data. This information is saved in the logger.

Programming \ Data objects

Programming \ Data objects \ Simple data objects in B&R data object syntax

Programming \ Libraries \ Data access and data storage \ DataObj

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

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

Using the DataObj library

1) Create a data object.

   Create the data object "param" in Automation Studio's logical view.

2) Enter the data in the format of the structure "recipe_typ".

   Using the Automation Studio online help documentation for support, enter the data into the data object in the format of the structure "recipe_typ".

3) Read out the contents.

   Use the function blocks "DatObjInfo()" and "DatObjRead()" to read the data from the data object.

4) Analyze the data.

   Determine if the data read back out matches the data that you entered into the data object. Don't forget about the alignment!

6.3 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 FileIO library makes it possible to access the internal file system, 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

Possibilities with the FileIO library

To access the file system, it is necessary for the storage location to have a symbolic location. This is also 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 or with the "DevLink()" function block.

Programming \ Libraries \ Data access and data storage \ FileIO

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

Using the FileIO library

1) Create a text file.

   Create a text file with Notepad and type anything into it.

2) Copy the text file to the CompactFlash card.

   Copy the file to the CompactFlash card. The CompactFlash card can be accessed by FTP as well as by a card reading device. In ArSim, the "C" drive is the hard drive on your computer.

3) Read out the contents of the file.

   Set up a state machine. Define an enumerated data type that maps out the step numbers of the state machine. Write a program that opens the file, reads out its contents and closes it again.

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

The CompactFlash card can be partitioned when it is created with the Automation Studio / Runtime Utility Center.

Diagnostics and service \ Service tool \ Runtime Utility Center \ Creating a list / data medium \ Creating a CompactFlash card

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 that works 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?

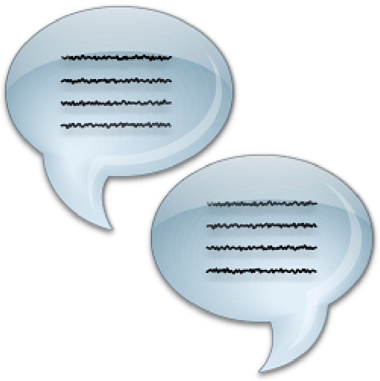
This list of questions provides clues to the issues that need to be cleared up before implementation can begin.

The Automation Studio help documentation provides an overview of the different types of communication available.

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

Communication

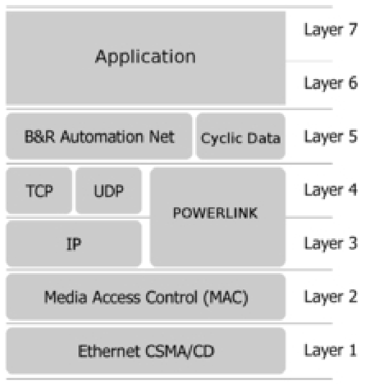
Programming \ Libraries \ Communication


Automation Studio help documentation - 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 and access methods. The topology indicates the type of connections used for communication.

Access methods refer to how different stations on the network coordinate with one another. 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.



OSI layer model

Frequently used topologies:

- Point-to-point
- Line
- Tree
- Star
- Bus
- Ring

Frequently used access methods:

- Master - Slave
- Token Ring
- CSMA/CA
- CSMA/CD
- Time slot method
- Client - Server

Common media:

- Serial RS232 / RS485 / RS422
- CAN
- Ethernet

Point-to-point

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.

Network

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 (e.g. CSMA / CD).

There are also access methods that define which stations on the network are allowed to send data, thus preventing collisions in the first place (e.g. CSMA / CA). POWERLINK uses a time slot method, for example, which also prevents collisions from occurring. Because of this and the amount of data that can be transferred, this type of transfer medium is real time capable.

Fieldbus systems

Fieldbus systems come into play when there is a device connected to a controller over a fieldbus. In Automation Studio, a fieldbus device can be inserted and configured directly on the fieldbus master. The description of the device is usually provided in a generic format (.eds, .gsd, .xdd file) that can be imported directly into Automation Studio.

Communication \ Fieldbus systems

7.2 Protocols

The communication protocol defines how data should be transferred from A to B. The receiver must know how the protocol is structured in order to be able to handle the data and determine what should be done with it. Data must be transferred in the correct format as well, i.e. both sides – the sender and the receiver – must know the data format that is being used.

Elements of a protocol include the following:

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

The protocol being used is not necessarily tied to a specific transmission technology. For example, CANopen doesn't always have to have something to do with CAN.

CANopen mechanisms are fully integrated into POWERLINK, for example.

Communication

7.3 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 included as constants and described in the help documentation for the respective library.

With the communication function blocks, an interface usually needs to be opened or initialized. In addition, a connection to the other station(s) needs to be established. If communication is no longer necessary, then the connection is broken and/or the interface closed.

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 and monitor a Transmission Control Protocol (TCP) connection. 3rd-party devices such as controllers and IP cameras can also be connected to a B&R controller.

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 a sample project.

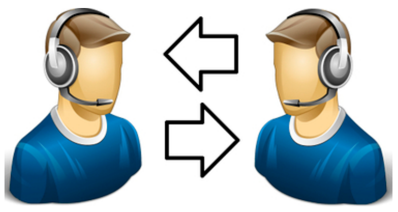
   Import the sample project "LibAsTCP1_ST".

3) Establish the connection to your communication partner.

4) Test the communication.

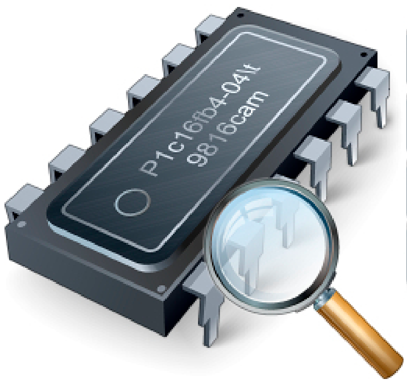
5) Test scenarios in which communication fails.

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


TCP communication

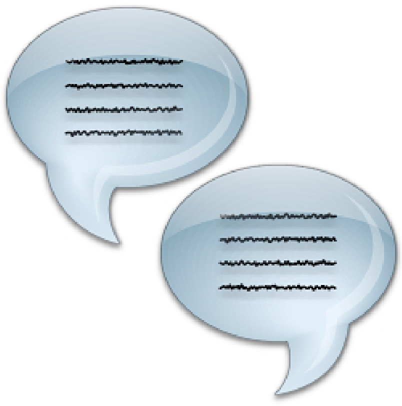
8 SUMMARY

Anyone who is responsible for designing a system is constantly confronted with data. How this data is organized on the system itself is extremely important. Basic data types, arrays and structures are available in this regard. They form the foundation for how data can be stored and transported.



Memory and variables

Data processing and storage


Communication and data exchange

Data can also be stored in files, memory areas or data objects. In these cases, the data format used to store the data needs to be defined. The data format, along with the protocol, also needs to be defined when transferring data. Both stations – the sender and the receiver – need to know what these are.

Data that is stored or transported from the cyclic program has to remain consistent at all times.

TRAINING MODULES

TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM230 – Structured Software Generation
TM240 – Ladder Diagram
TM241 – Function Block Diagram (FBD)
TM246 – Structured Text
TM250 – Memory Management and Data Storage
TM261 – Closed Loop Control with LOOPCONR
TM440 – ASiM Basic Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors
TM500 – Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER
TM540 – ASiST SafeMC
TM600 – The Basics of Visualization
TM630 – Visualization Programming Guide
TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX