

# Contents

<b>11. Data Blocks.....</b>	<b>11-2</b>
11.1. Data Blocks and their Usage .....	11-3
11.2. Meaning of Variables and Data Types.....	11-4
11.2.1. Overview: Data Types in STEP 7 .....	11-5
11.2.1.1. Complex Data Types 1 .....	11-6
11.2.1.2. Complex Data Types 2 .....	11-8
11.3. Creating a Global Data Block.....	11-9
11.3.1. Editing a Data Block.....	11-10
11.3.2. Default, Start and Monitor Values .....	11-11
11.3.3. Retentiveness, Download DB into the CPU / Upload from the CPU .....	11-12
11.3.3.1. Downloading Changed Data Blocks into the CPU.....	11-13
11.3.4. Snapshot, Setpoint, Start Value.....	11-14
11.3.4.1. Initializing Setpoints Online.....	11-15
11.3.4.2. Changing the Snapshot / Start Value of Several / All Data Blocks.....	11-16
11.3.5. Copy & Paste from / to Microsoft Excel .....	11-17
11.4. Exercise 1: Creating Data Block "DB_OP" .....	11-18
11.4.1. Exercise 2: Adjusting "FB_Count" and Updating the Call.....	11-19
11.4.2. Exercise 3: Using DB Variables as Actual Parameters .....	11-20
11.5. Additional Information .....	11-21
11.5.1. Example of a Variable of the Data Type ARRAY.....	11-22
11.5.2. Example of a Variable of the Data Type STRUCTURE.....	11-23
11.5.3. PLC Data Type.....	11-24
11.5.4. Functions RD_SYS_T and RD_LOC_T .....	11-25

## 11. Data Blocks

**At the end of the chapter the participant will ...**



- ... understand the purpose of global data blocks
- ... be familiar with complex data types
- ... be able to create and edit global data blocks and use them in the program
- ... be familiar with the possibilities for addressing data block variables
- ... be able to monitor and initialize a data block
- ... be familiar with the difference between default values, start values, monitoring values, setpoints and snapshots

## 11.1. Data Blocks and their Usage

The screenshot displays two windows from the SIMATIC Manager. The left window shows the Ladder Logic (LAD) editor for a function block (FB) named 'CountADD [FB19]'. It contains three networks: Network 1 (Reset Counter), Network 2 (Counting the transported parts), and Network 3 (Actual equal setpoint ??). Network 2 uses an 'ADD UInt' block with inputs from '%M19.4', '%Q0.1', and 'M\_AuxCount(19)'. The output of the 'ADD' block is connected to 'DB\_OP', Act\_No'. Network 3 compares 'DB\_OP', Act\_No' with 'DB\_OP', Setp\_No' using an equals sign and connects the result to 'M\_ActSetp'. The right window shows the SIMATIC HMI interface for 'SITRAIN'. It features a 'Plant ON' button, a 'Plant OFF' button, and a 'Jog left' button. A central display shows 'Setpoint quantity' as 3 and 'Actual quantity of parts' as 1. Below this is a diagram of a conveyor system. A red dashed line connects the 'DB\_OP', Act\_No' variable in the LAD editor to the 'Act\_No' variable in the HMI interface table.

	Name	Data type	Start value	Monitor value
1	Static			
2	Operation_OFF	Bool	false	FALSE
3	Operation_ON	Bool	false	FALSE
4	Jog_Right	Bool	false	FALSE
5	Jog_Left	Bool	false	FALSE
6	Ack_Fault	Bool	false	FALSE
7	Act_No	UInt	0	1
8	Setp_No	UInt	6	3

Data blocks contain variables for storing user data and accordingly occupy memory space in the work memory of the CPU.

### Area of Application

You can use data blocks in different ways, depending on their contents. You differentiate between:

- Global data blocks: These contain information that all the code (logic) blocks in the user program can access. Often, global data blocks are also used as an interface to HMI devices (operating and monitoring devices).
- Instance data blocks: These are always assigned to a particular FB. The data of these instance DBs should only be processed by the associated FB.

### Creating DBs

Global DBs are created either with the Program Editor or according to a previously created PLC data type. Instance data blocks are generated by the Editor according to a function block.

## 11.2. Meaning of Variables and Data Types

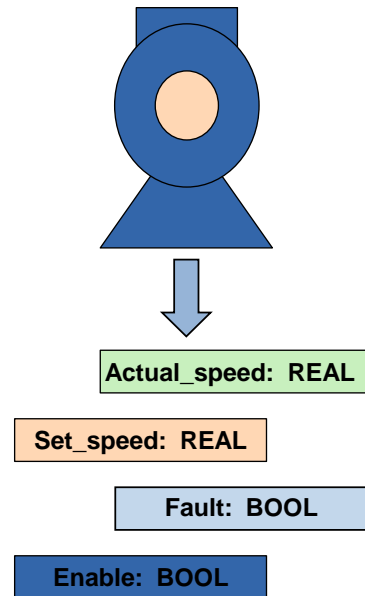
**Variables represent the abstraction of reality and permit you to save and later continue to process values.**

**By declaring a variable, the following properties are defined:**

- Symbolic name
- Memory area
- Validity range
- Data type

**The data type establishes:**

- The possible value range (e.g. INT: -32 768 to +32 767)
- The permitted instructions (e.g. math instructions: +!, -!)
- How the bits in the memory are to be interpreted (integer; hexadecimal number; floating-point number; etc.)



### The Meaning of Variables

Next to commands, variables are the most important elements of a programming system. Their task is to save values in a program so that they can be further processed at a later time. The value of a variable can be saved "anywhere" in the PLC memory.

The data represents an abstraction of reality in which irrelevant properties of objects are ignored.

### Data Types

It is often quite difficult to decide how data is to be represented and the available possibilities quite often restrict the choice. On the one hand, the object properties the data describe must be correctly reflected. On the other hand, it must also be possible to carry out the instructions necessary for process with the data.

The data type determines which values are accepted by data and which instructions can be carried out with these values.

The data type uniquely defines

- the possible value range
- the permitted instructions
- how the bit pattern is to be interpreted

### 11.2.1. Overview: Data Types in STEP 7

	Type	Data types
<b>Elementary Data types</b>	Binary number	BOOL
	Bit sequences	BYTE; WORD; DWORD; LWORD
	Integers	SINT; USINT; INT; UNIT; DINT; UDINT; LINT ULINT
	Floating-point numbers	REAL; LREAL
	Timers	S5TIME; TIME; LTIME
	Date, Time-of-day	DATE; TIME_OF_DAY; LTIME_OF_DAY; LDT(DATE_AND_LTIME);
	Characters	CHAR; WCHAR
<b>Complex Data types</b>	Date, Time-of-day	DT(DATE_AND_TIME); DTL;
	Character string	STRING; WSTRING
	Array	ARRAY [...] of <Datatype>
	Anonymous Structure	STRUCT
	User-defined	PLC-data type (User Defined Data Type)









#### Elementary Data Types

Elementary data types are predefined in accordance with IEC 61131-3. They always have a length less than or equal to 64 bits.

#### Complex Data Types

Complex data types contain data structures that can be made up of elementary and/or complex data types. Complex data types can be used for the declaration of variables only in global data blocks and within blocks for the declaration of local variables (TEMP, STAT) as well as parameters (IN, OUT and INOUT).

### 11.2.1.1. Complex Data Types 1

Data type	Length (Bits)	S7-1200	S7-1500	Example
<b>DT</b> (DATE_AND_TIME)	64			<b>DT#2008-10-25-08:12:34.567</b>
<b>DTL</b>	96			<b>DTL#1976-12-16-20:30:20.250</b>
<b>STRING</b>	8 * (Number of characters+2)			<b>'This is a String'</b> max. 254 characters in ASCII format
<b>WSTRING</b> (Wide Character String)	16 * (Number of characters+2)			<b>WSTRING#'STRING in UNICODE format'</b> Up to 16382 characters in Unicode format

#### DT

The data type DATE\_AND\_TIME represents a point in time consisting of the date and the time-of-day. Instead of DATE\_AND\_TIME, the abbreviation DT can also be used.

Byte	Contents	Range of values
0	Year	0 to 99 (Years 1990 to 2089) BCD#90 = 1990 ... BCD#0 = 2000 ... BCD#89 = 2089
1	Month	BCD#0 to BCD#12
2	Day	BCD#0 to BCD#31
3	Hour	BCD#0 to BCD#23
4	Minute	BCD#0 to BCD#59
5	Second	BCD#0 to BCD#59
6	The two most significant digits of MSEC	BCD#0 to BCD#999
7 (4MSB) <sup>1)</sup>	The least significant digit of MSEC	BCD#0 to BCD#9
7 (4MSB) <sup>2)</sup>	Weekday	BCD#1 to BCD#7 BCD#1 = Sunday ... BCD#7 = Saturday

**DTL**

The data type DTL has a length of 12 bytes and, like LDT, stores information on date and time-of-day precise to the nanosecond since 1.1.1970, only in a pre-defined structure.

**Advantage:** the individual values (day, hour, etc.) are easier to read out.

Byte	Component	Data type	Range of values
0 - 1	Year	UINT	1970 to 2554
2	Month	USINT	1 to 12
3	Day	USINT	1 to 31
4	Weekday	USINT	1 (Sunday) to 7 (Saturday) The weekday is not considered in the value entry.
5	Hour	USINT	0 to 23
6	Minute	USINT	0 to 59
7	Second	USINT	0 to 59
8 -11	Nanosecond	UDINT	0 to 999 999 999

**STRING**

The data type STRING stores several ASCII characters of the data type CHAR in a character string with a maximum of 254 characters.

**WSTRING**

The data type WSTRING (Wide Character String) stores several Unicode characters of the data type WCHAR in a character string with a maximum of 16382 characters.

### 11.2.1.2. Complex Data Types 2

Data type	Length (Bits)	S7-1200	S7-1500	Example	
ARRAY	User-defined	✓	✓	Measured values: ARRAY[1..20] of INT;	
STRUCT	User-defined	✓	✓	<b>Motor: STRUCT</b> ACT-Speed : REAL; Set-Speed : UINT; Fault : BOOL; Enable : BOOL; <b>END_STRUCT</b>	
PLC-Data type (UDT)	User-defined	✓	✓	Define (in the PLC-data types folder)	Use (in data blocks and interfaces)
				<b>PLC-DT1 : STRUCT</b> Speed : REAL; Enable : BOOL; Fault : BOOL; <b>END_STRUCT</b>	<b>Motor1 : PLC-DT1;</b> <b>Motor2 : PLC-DT1;</b> <b>Motor3 : PLC-DT1;</b> :

Arrays and Structures consist of groups of elementary or complex data types.

They enable you to create data types suitable for your task with which you can structure large quantities of data and process it symbolically.

#### ARRAY and STRUCT

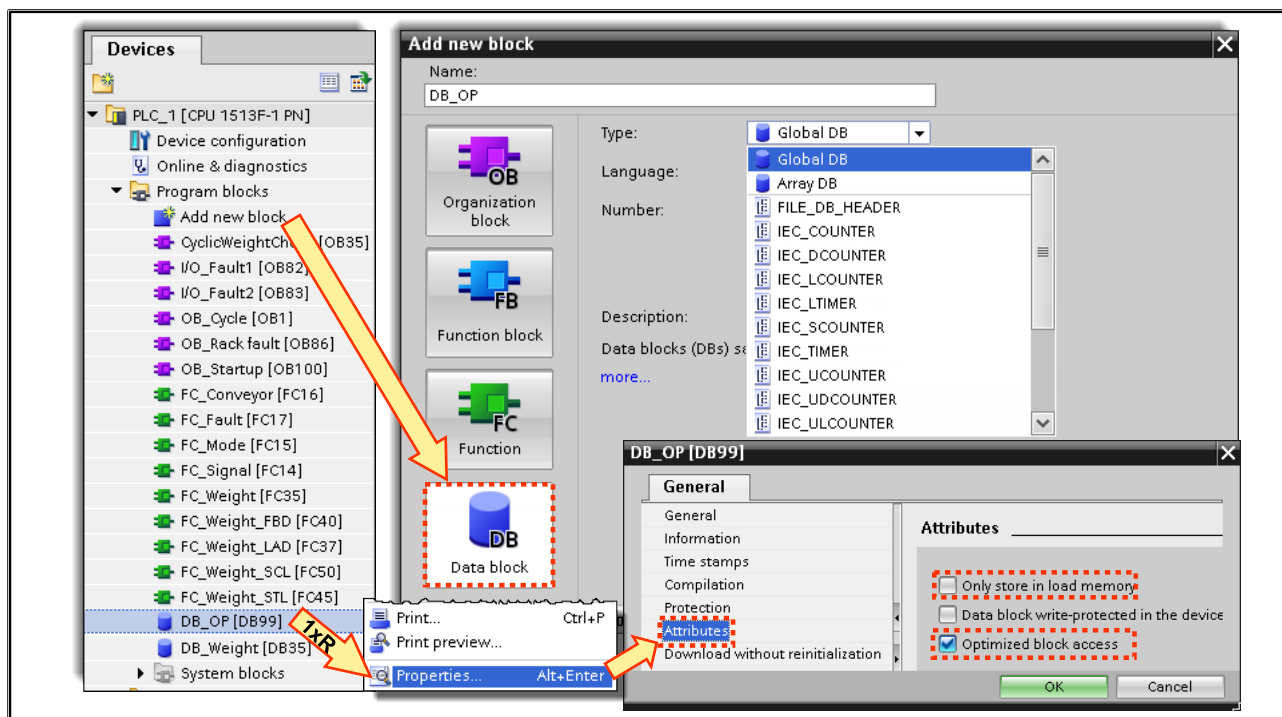
ARRAYs and Structures can only be declared within global data blocks and as parameters or local variables of code blocks.

#### PLC-Data Type (User-defined Data Type)

User-defined data types represent self-defined structures. This structure is stored in the PLC data types and can be used as a "template" in another variable's data type.



### 11.3. Creating a Global Data Block



#### Creating a Global Data Block

(Global) data blocks are created in exactly the same way as code (logic) blocks.

In creating a data block you can select of which type the data block is to be or for what purpose it is to be used:

- Global-DB for saving global data or for creating global variables
- Instance-DB or "private memory" for a user function block or a particular "instruction", behind which a function block (FB) is also ultimately hidden

#### "Optimized Block Access" Attribute

Data blocks can be created with the attribute "Optimized block access":

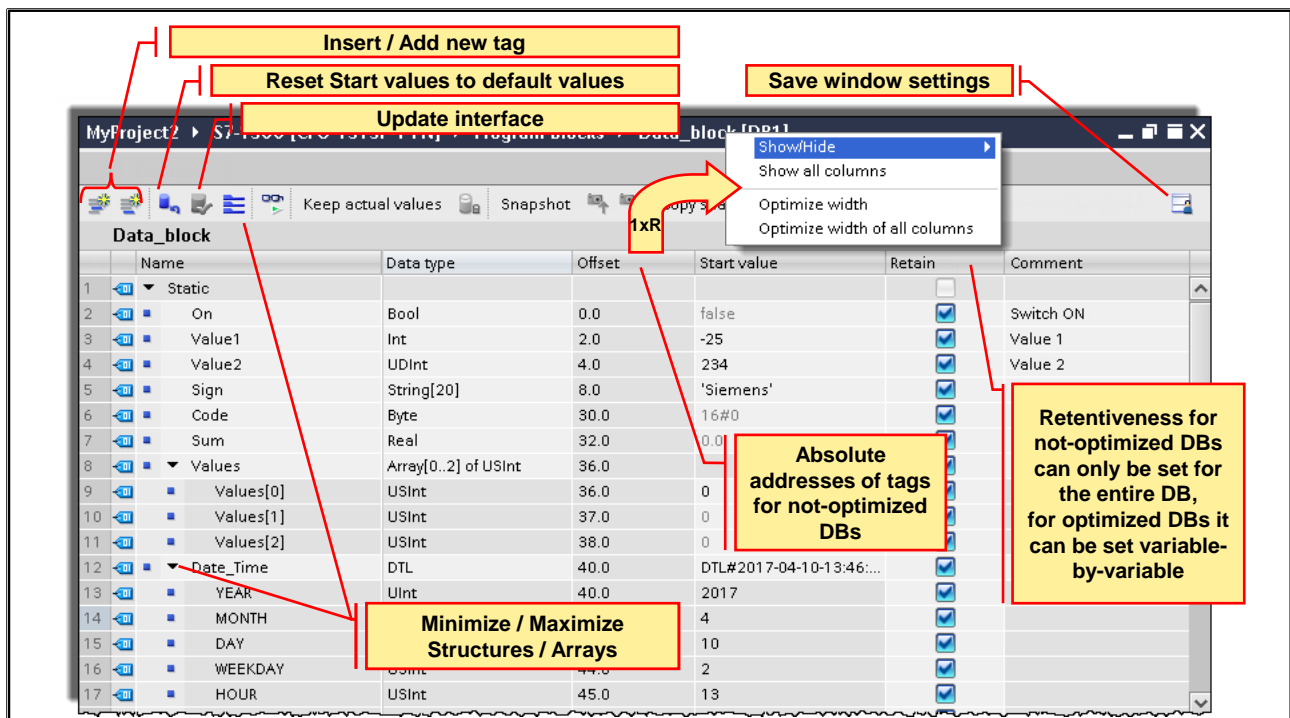
- In S7-1200, these data blocks are created memory-optimized, that is, the variables are stored in sequence so that fill bytes, through the changing, successive creating of variables of the dimensions bit, byte word and double-word, are no longer required.
- In S7-1500, these data blocks are created access-optimized taking the S7 machine code into consideration, so that – regardless of the memory requirements of the variables – the access times to these are minimized.

**Optimized Block Access see:** TIA Portal Information Center > Documentation > Manuals >Control Technology > Programming Guideline for an optimal programming of SIMATIC S7-1200/1500 controllers

#### "Only Store in Load Memory" Attribute

This attribute means that in downloading into the CPU, the data block is only loaded into the load memory of the CPU and from there is not automatically adopted in the work memory.

### 11.3.1. Editing a Data Block



#### Offset

The offset denotes the absolute address of a variable within a DB. In the STEP 7 program, the use of the symbolic address or name is preferable since it is easier to read and less prone to errors than the absolute addressing.

The absolute addresses of variables within the data block are not displayed for optimized blocks and can also not be read out.

#### Retentiveness

For not-optimized blocks, the retentive behavior cannot be defined for the individual variables, but always only for all variables or the entire data block.

When "Retain" is activated, the monitoring values are retained in the CPU until the data block is initialized.

When "Retain" is not activated, the monitoring values in the work memory are overwritten with the start values from the load memory after every STOP - RUN - transition (by PG, mode selector switch or Power OFF->ON) of the CPU.

### 11.3.2. Default, Start and Monitor Values

Current value of variable in the CPU (online)

Snapshot of current value (offline)

Start value of variable

Default start value

Name	Data type	Offset	Default value	Start value	Snapshot	Monitor value	Setpoint
Static							
On	Bool	0.0	false	false	FALSE	TRUE	
Value1	Int	2.0	0	-25	456	543	
Value2	UDInt	4.0	0	234	0	0	
Sign	String[20]	8.0	"	'Siemens'	'Siemens'	'Siemens'	
Code	Byte	30.0	16#0	16#0	16#3F	16#23	
Sum	Real	32.0	0.0	0.0	0.0	345.23	
Values	Array[0..2] of USInt	36.0					
Values[0]	USInt	36.0	0	0	234	234	
Values[1]	USInt	37.0	0	0	0	23	
Values[2]	USInt	38.0	0	0	0	45	
Date_Time	DTL	40.0	DTL#1970-01...	DTL#2017-04-10-1...	DTL#2017-04-10-15-0...	DTL#2017-04-12-13:...	
YEAR	UInt	40.0	1970	2017	2017	2017	
MONTH	USInt	42.0	1	4	4	4	

#### Default Value

Default values cannot be edited within global data blocks, only within PLC data types (the default values of structure elements) and within FBs (the default values of parameters and static variables).

If, within a global DB, a variable of the data type Structure is declared according to the PLC data type x, the default values edited in the PLC data type are then displayed (as not editable) in the data block and adopted as (editable) start values.

Within an FB, their default values also have to be specified in the declaration of parameters and static variables. In the instance DBs of the FB, these default values are then displayed (as not editable) and adopted as (editable) start values.

#### Start Value

In declaring a variable, the default value of a variable is automatically adopted as a start value; however, this value can be overwritten at any time and downloaded into the controller. After a data block was loaded into the CPU the first time, the CPU starts the program execution with exactly this value. For all non-retentive variables, this value is written in the Monitor value with every STOP-RUN-transition or for with an initialization; for retentive variables, only for a re-initialization.

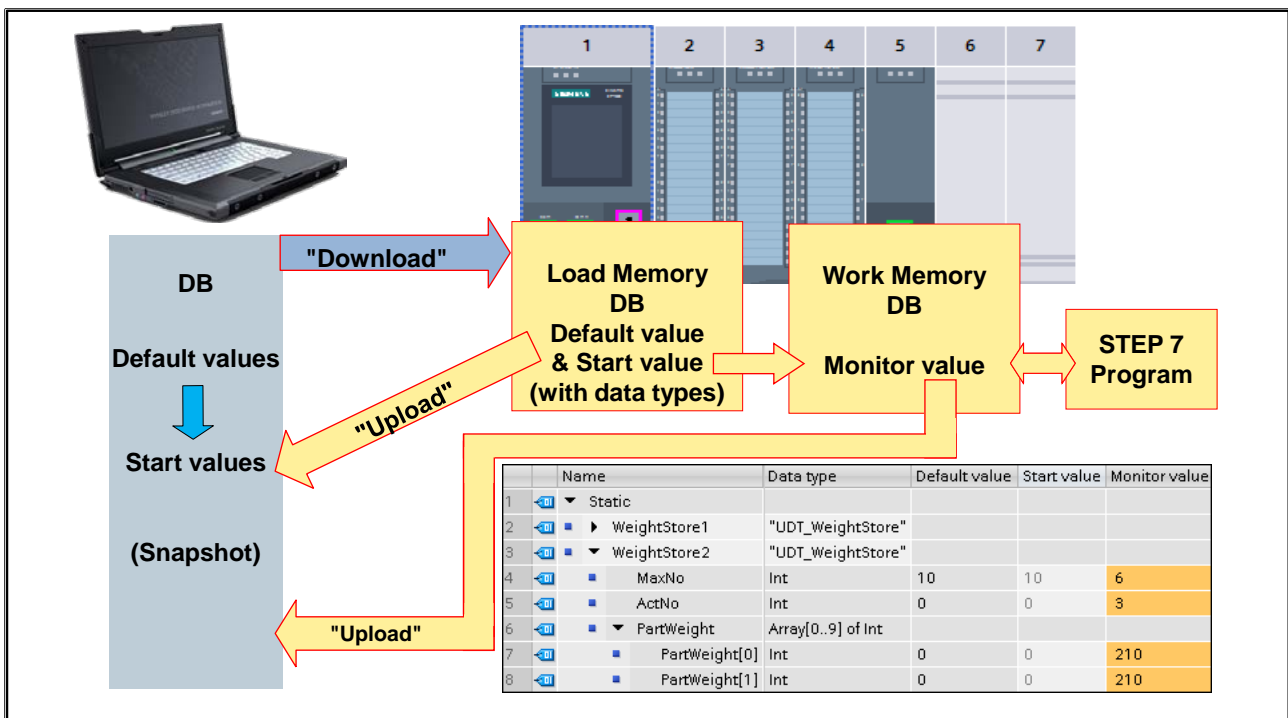
#### Monitoring Value

The monitoring value of a variable is the current value that the variable presently has in the work memory of the CPU.

#### Snapshot

The value "Snapshot" is a monitoring value for an already passed point in time x, at which the current monitoring values are read out of the CPU and stored offline as the values of the "Snapshot".

### 11.3.3. Retentiveness, Download DB into the CPU / Upload from the CPU



#### "Download" (to Device)

In downloading a data block from the PG into the CPU, the default values and start values as well as the data types of the variables are transferred into the load memory of the CPU. From the load memory, the start values are then automatically applied as monitoring values in the work memory of the CPU. The STEP 7 program works with the monitoring values of the work memory.

#### "Upload" (from Device)

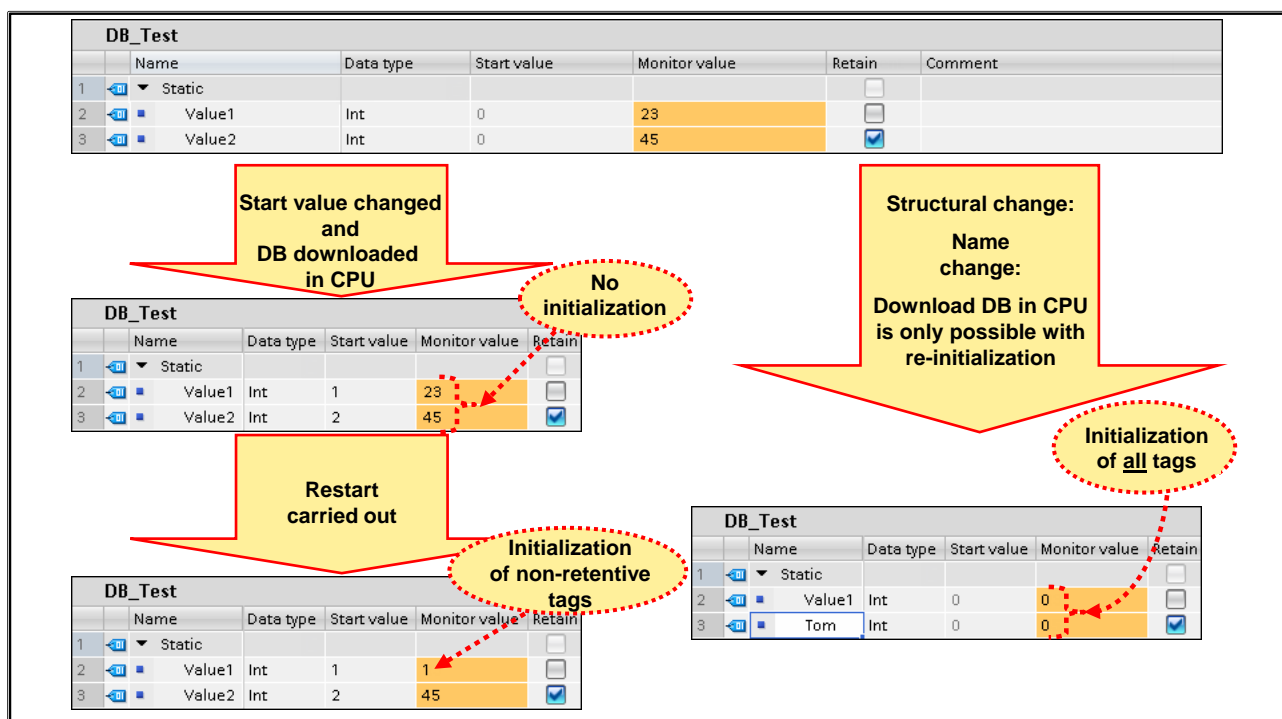
In uploading a data block from the CPU into the PG, the start values from the load memory and the current monitoring values from the work memory are transferred into the PG. In doing so, the "snapshot" values stored offline are overwritten with the monitoring values loaded from the CPU. The values from this "snapshot" can then be applied offline in the project as new "start values".

#### Retentiveness (Retain)

With retentive variables, the "monitor values" (actual values) are retained after a CPU restart.

Non-retentive variables occupy work memory but no retentive memory and are thus reset to the start values from the load memory with every CPU restart (Power→Off→On or after every STOP - RUN).

### 11.3.3.1. Downloading Changed Data Blocks into the CPU



#### Downloading Data Blocks with Re-initialization

No matter whether a DB was created with standard or optimized block access, a change of start values is not a structural change. Therefore, it can be done and downloaded into the CPU without a re-initialization of the data block variables (tags) being necessary.

On the other hand, after structural changes, it is only possible to download the data block by re-initializing all tags.

Structural changes are:

- Name changes (since synonymous with deleting a variable (tag) and creating a new one)
- Changes to the retentive behavior
- Adding / removing tags

#### DB Re-initialization during Restart

No matter whether a DB is created with standard or optimized block access, all monitor values of non-retentive tags are overwritten with their start values when the CPU is restarted.

### 11.3.4. Snapshot, Setpoint, Start Value

Load Snapshot as Monitor values / Actual values

Snapshot of the Monitor values / Actual values

Insert new variable without losing actual values

Monitor On/Off (Block)

... Area\_1 ▶ PLC\_1 [CPU 1513F-1 PN] ▶ Program blocks ▶ Data\_block\_1 [DB1]

Keep actual values Snapshot Copy snapshots

Data\_block\_1 (snapshot created: 04.04.2017 08:17:57)

	Name	Data type	Start value	Snapshot	Monitor value	Setpoint
1	Static					
2	Value_1	Bool	True	FALSE	TRUE	<input checked="" type="checkbox"/>
3	Value_2	Int	10	22	22	<input type="checkbox"/>



Apply snapshots as start values (All or only setpoints)

Initialize variables (All or only setpoints)



Program blocks ▶ Data\_block\_1 [DB1]

Copy snapshots to start values Load start values as actual values



#### Snapshot

With the "Snapshot of monitor value"  function, the actual values of the online DB are stored in the offline DB. With the "Load snapshot as monitor value / actual value"  function, the values are once more loaded into the online DB.

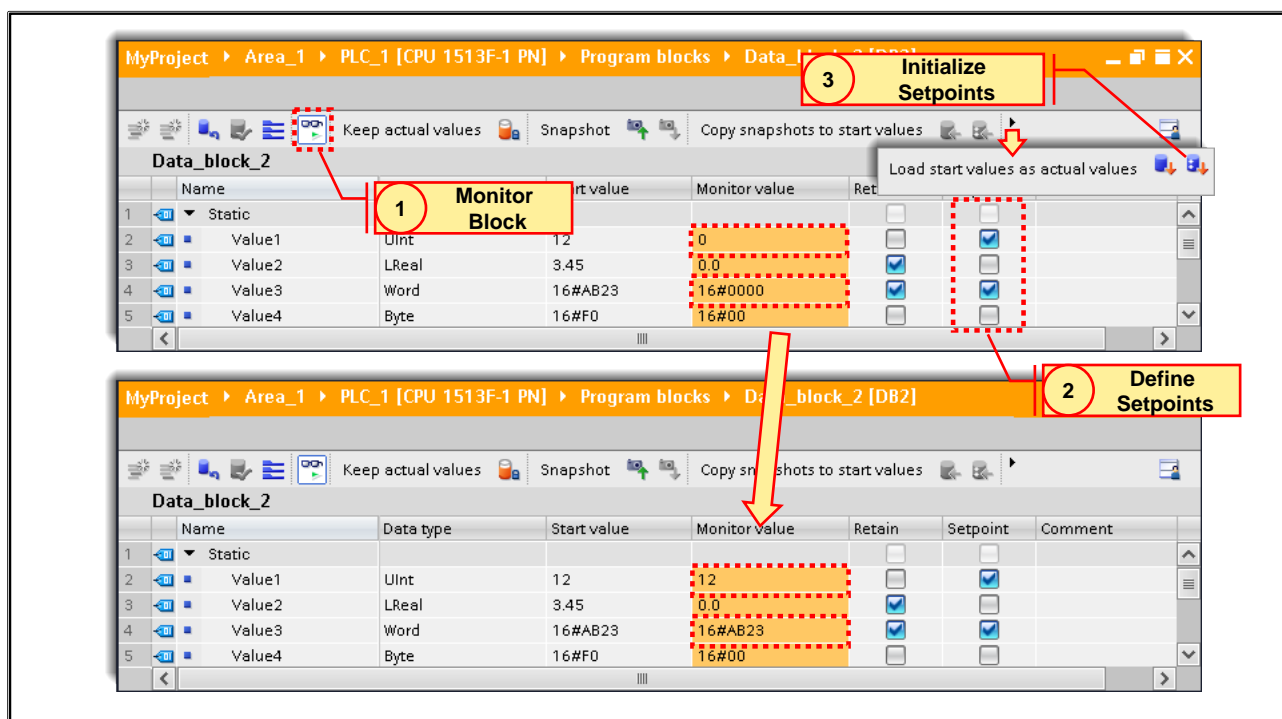
#### Copy Snapshot to Start Values

Saved values can be adopted in the Start value column by means of the  button (for all snapshots selected in the Setpoint column) or the  button (for all snapshots). The next time this DB is transferred, these values are applied as start value (no structural change of the DB -> a download without re-initialization is possible).

#### Adjusting Actual Values

Furthermore, the entire DB can be initialized with the  button and all values selected in the Setpoint column can be initialized with the  button.

### 11.3.4.1. Initializing Setpoints Online



#### Initializing Setpoints in the Online Program

All variables that are checkmarked as "Setpoint" can be initialized online in the CPU. Online, the monitor values are overwritten with the start values. The CPU remains in "RUN". The changed monitor values are applied once at the next cycle control point. This applies for retentive as well as for non-retentive variables. The program execution then continues with the new variable values. Prerequisite is an online connection to the CPU, the structure of the data block is identical online and offline and one or more variables are checkmarked as "Setpoint".

#### What to Do:

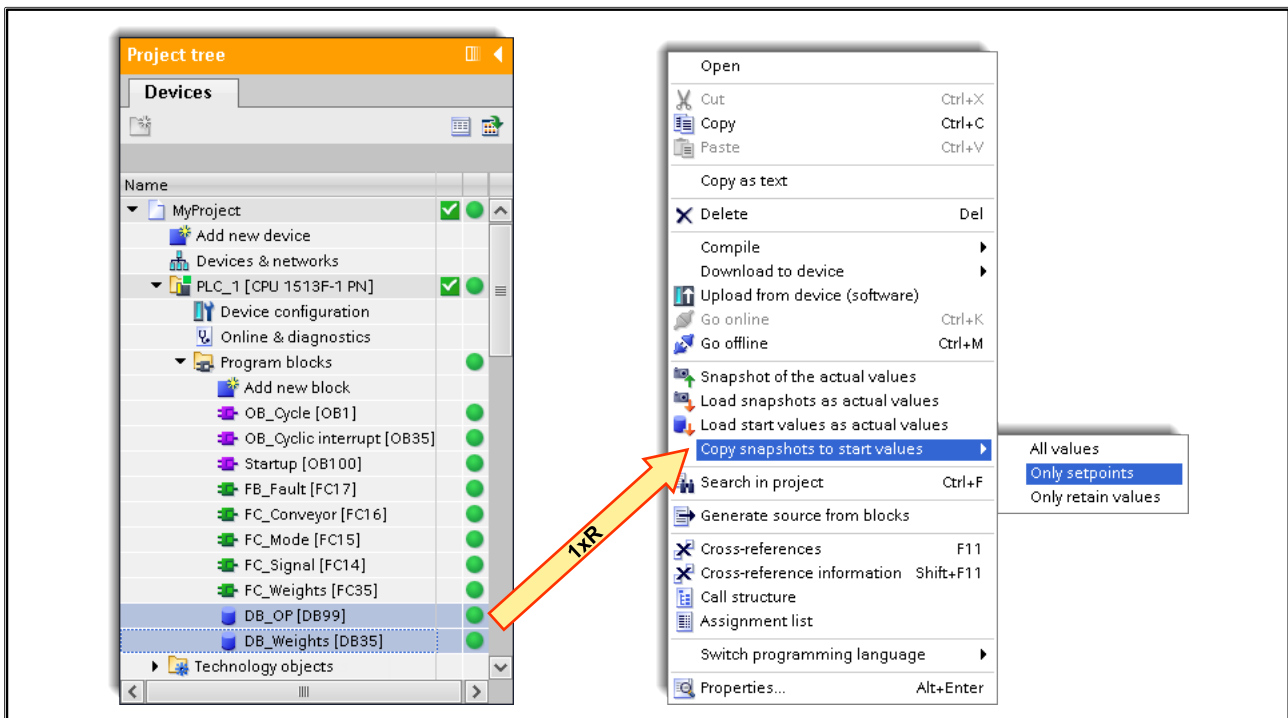
1. Open the global data block or instance data block and establish an online connection by monitoring it.
2. In the "Setpoint" column, checkmark the variables whose monitor values are to be overwritten with the start values in the CPU.
3. Click on the "Initialize Setpoints" button in order to initialize the variables checked as "Setpoints".

#### Note:

For global data blocks, the "Setpoints" checkmark can only be set and reset for those variables which have not been declared according to PLC-data type, without having to reload the block. For variables that are declared according to PLC-data type, the "Setpoint" checkmark must be made in the PLC-data type and then the DB has to be reloaded.

For instance data blocks, only the static variables can be initialized with "Initialize Setpoints".

### 11.3.4.2. Changing the Snapshot / Start Value of Several / All Data Blocks



#### Snapshot of Several / All Data Blocks

Just as a snapshot can be made, written back, the DB initialized and snapshots copied into the start values for an individual data block, this can also be done for several or all blocks.



### 11.3.5. Copy & Paste from / to Microsoft Excel

The screenshot illustrates the process of copying data from a SIMATIC TIA Portal data block to Microsoft Excel. The 'Data\_block\_2' table is shown with the following data:

	Name	Data type	Start value	Retain	Setpoint	Comment
1	Static					
2	Value1	UInt	12			
3	Value2	LReal	16497.0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
4	Value3	Word			<input checked="" type="checkbox"/>	
5	Value4	Byte			<input checked="" type="checkbox"/>	

A context menu is open over the table, with the 'Copy' option selected. A red arrow points from the 'Copy' option to an Excel spreadsheet. The Excel spreadsheet shows the copied data in the following table:

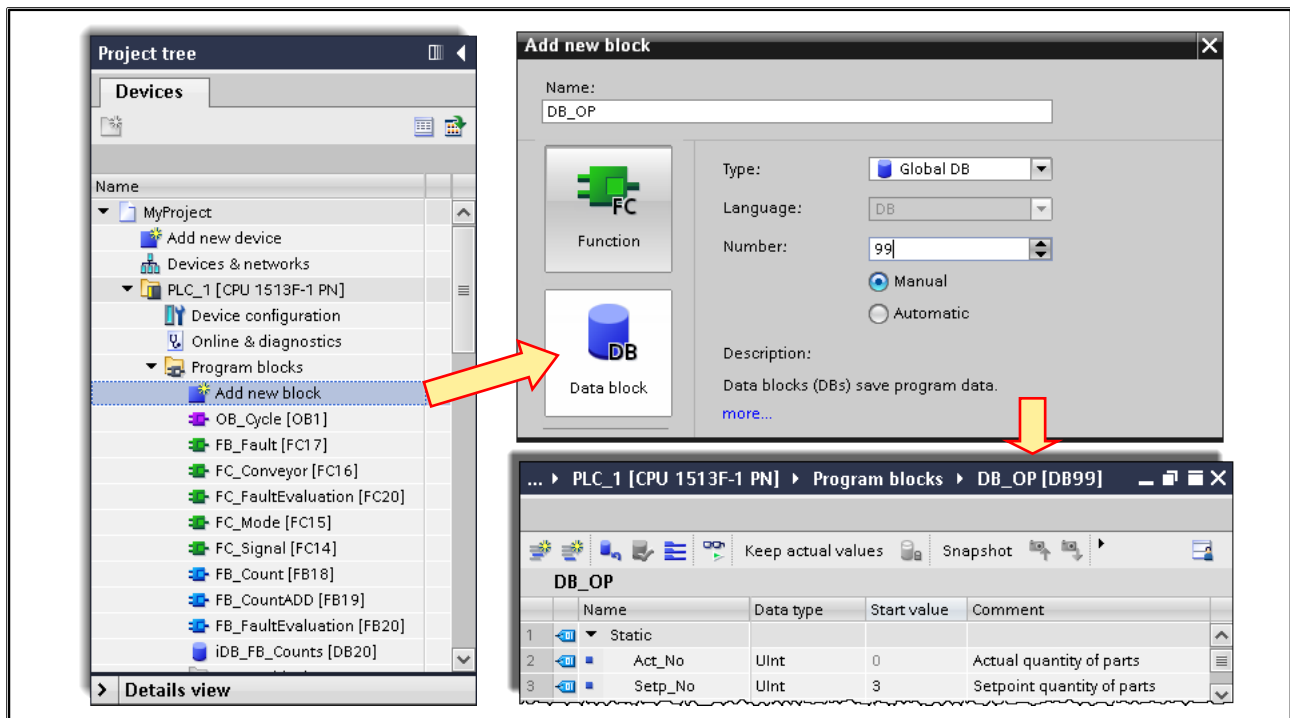
	A	B	C	D	E	F
1	Value1	UInt	12	False	True	
2	Value2	LReal	16497.0	True	False	
3	Value3	Word	16#AB23	True	True	
4	Value4	Byte	16#F0	False	False	

A yellow box with the text 'Copy & Paste to Excel' is located below the context menu.

#### Copy & Paste from and to Excel

The Windows Copy & Paste function can be used to easily copy individual or several variables from a data block to Excel to further process it/them there and then to copy it/them back from Excel to the data block.

## 11.4. Exercise 1: Creating Data Block "DB\_OP"



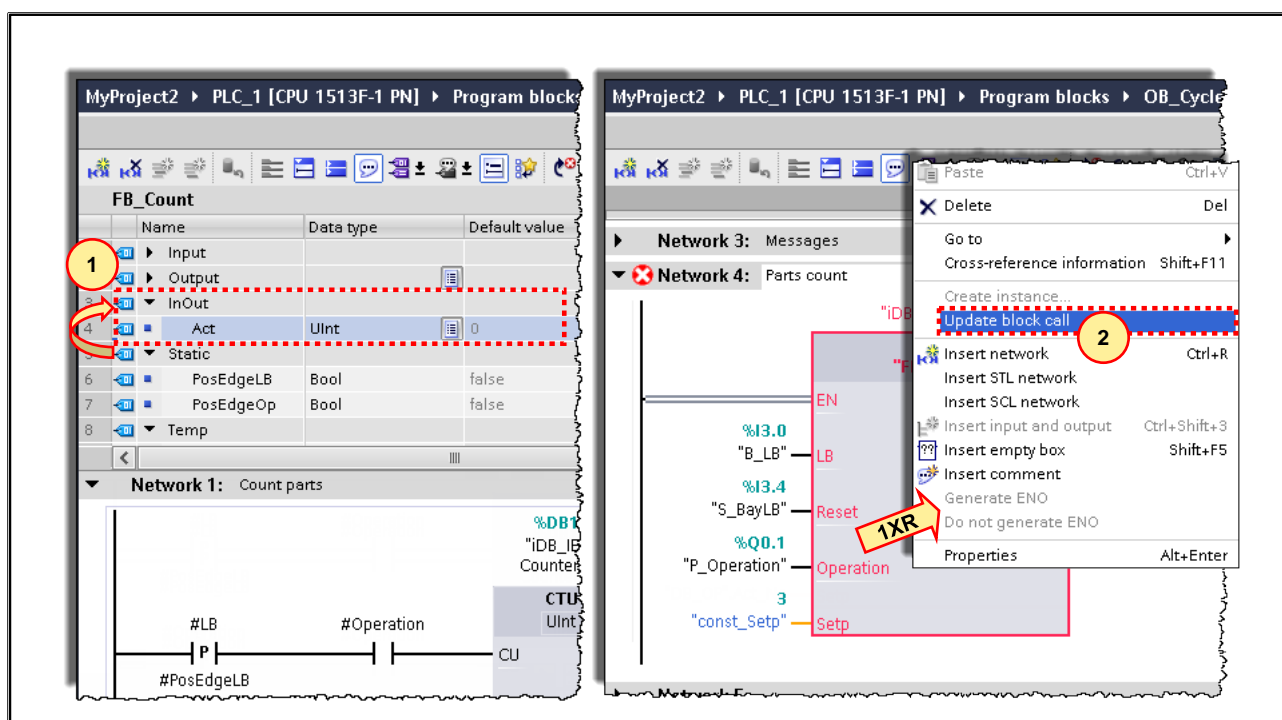
### Task:

You are to create the data block "DB\_OP" with the variables (tags) shown in the picture. The variables are to be used in the STEP 7 program and are also to serve as an interface to the touchpanel.

### What to Do:

1. Create the new "DB\_OP" as a Global DB.
2. Declare the variables as shown in the picture above. Give the variable "SetpNo" the Start value 3.
3. Save your project.

### 11.4.1. Exercise 2: Adjusting "FB\_Count" and Updating the Call



#### Function Up Until Now:

The transported parts are counted in "FB\_Count" as soon as they have passed the light barrier.

The current count, that is, the actual quantity is stored in the static variable #Act and the setpoint quantity is preset with constant 3 in the program.

If the actual quantity has reached the setpoint quantity, it is indicated on the conveyor model indicator light "P\_BayLB" (Q3.4) with a 1Hz flashing light and no new transport sequence can be started. By acknowledging this signal with the conveyor model pushbutton "S\_BayLB" (I 3.4), the static variable #Act is overwritten with 0 and further transport sequences can be started.

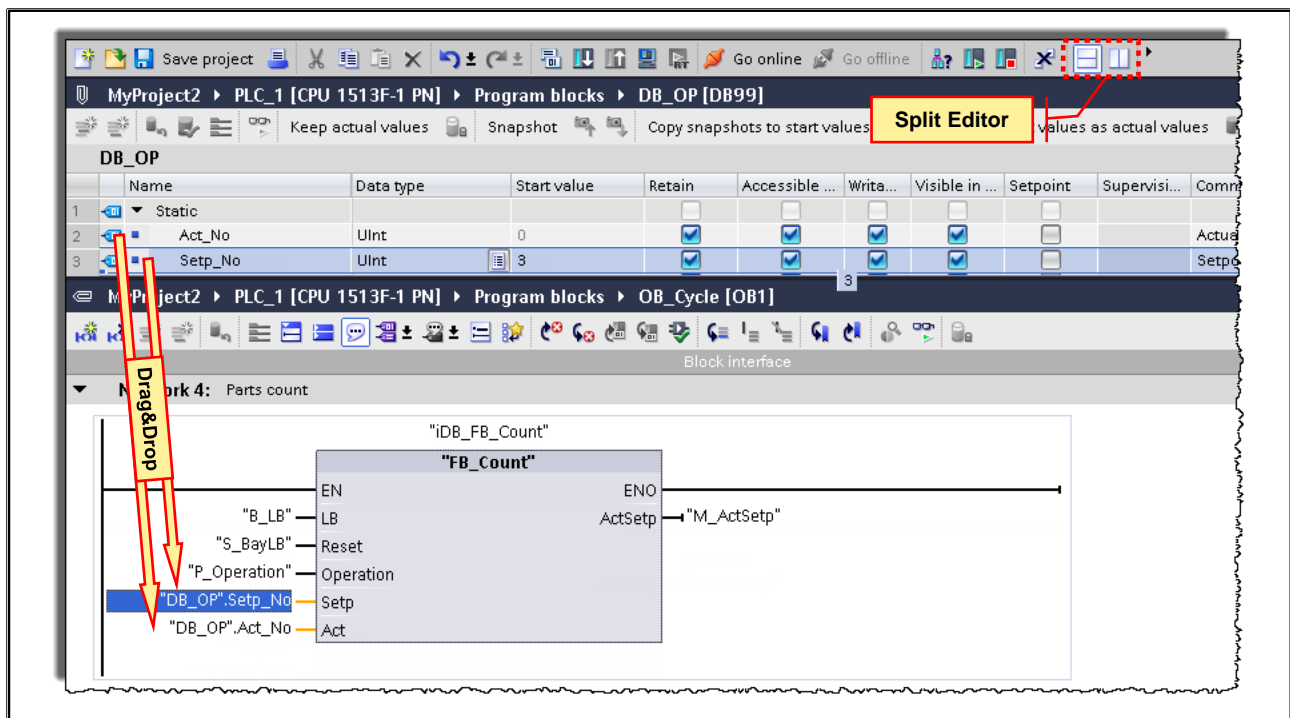
#### Task

The function of "FB\_Count" remains unchanged, however, the actual quantity is no longer to be stored in the static variable #Act and the setpoint quantity is no longer to be preset with constant 3. Instead, the data block variables (tags) "DB\_OP".ActNo and "DB\_OP".SetpNo are to be used.

#### What to Do:

1. Change the static variable #Act into an InOut parameter of the same data type (UINT) and save the function block.
2. Open the Organization block "OB\_Cycle".
3. Update the call of "FB\_Count". For this, open the context menu by right-clicking on the call of "FB\_Count" and activating the function "Update block call".
4. Confirm the dialog "Interface synchronization" with "OK".

### 11.4.2. Exercise 3: Using DB Variables as Actual Parameters



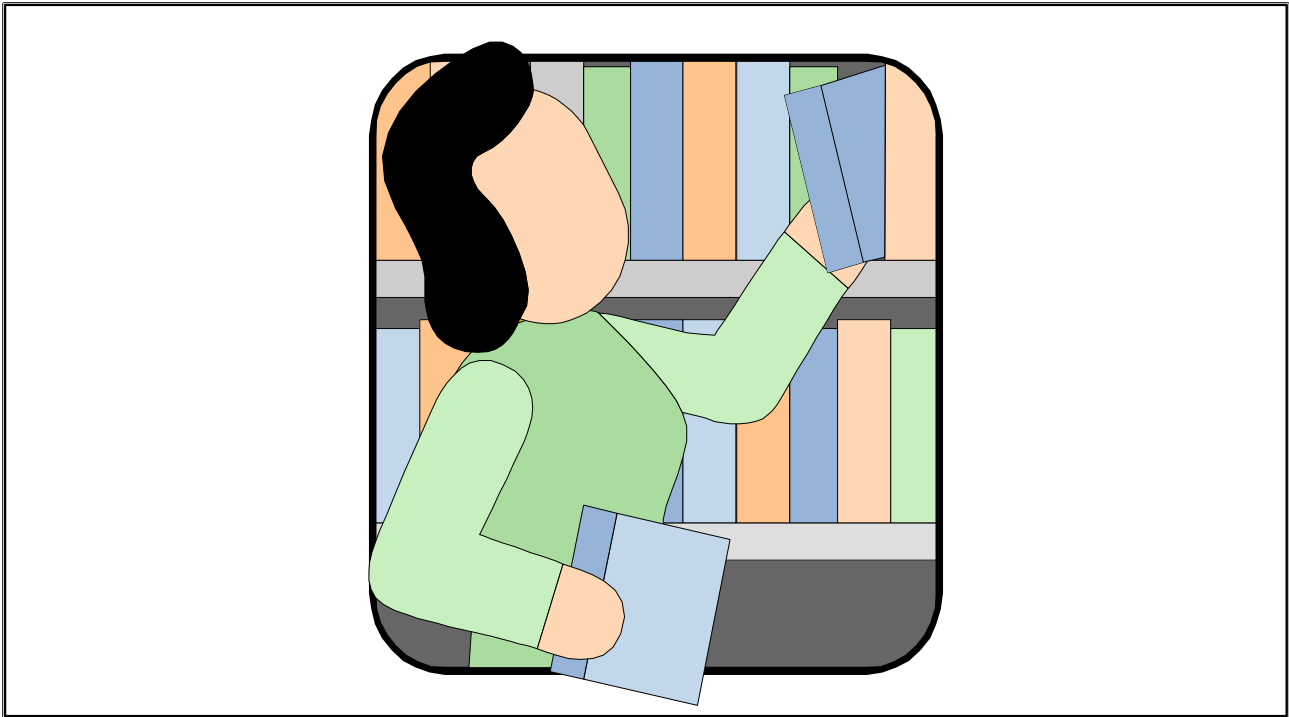
#### Task

You are to supply the formal parameters #Setp and #Act with the data block variables "ActNo" and "SetpNo" of the data block "DB\_OP".

#### What to Do:

1. Split the working area using the "Split Editor" button.
2. In one area open the data block "DB\_OP" and in the other the organization block "OB\_Cycle".
3. Supply the formal parameters #Setp and #Act with the data block variables "ActNo" and "SetpNo" of the data block "DB\_OP" using drag & drop.
4. Compile and save your project.
5. Monitor the "DB\_OP" data block while you transport parts in Automatic mode and increase the ACT quantity.

## 11.5. Additional Information



### **Note**

The following pages contain either further information or are for reference to complete a topic. For more in-depth study we offer advanced courses and self-learning mediums.

### 11.5.1. Example of a Variable of the Data Type ARRAY

Array [0..9] of LREAL with the name "Value"  
(10 elements of the data type LREAL)

	Name	Data type	Start value	Retain	Setpoint	Comment
1	Static					
2	Value	Array[0..9] of LReal				Measured values
3	Value[0]	LReal	0.0			
4	Value[1]	LReal	0.0			
5	Value[2]	LReal	0.0			
6	Value[3]	LReal	0.0			
7	Value[4]	LReal	0.0			
8	Value[5]	LReal	0.0			
9	Value[6]	LReal	0.0			
10	Value[7]	LReal	0.0			
11	Value[8]	LReal	0.0			
12	Value[9]	LReal	0.0			

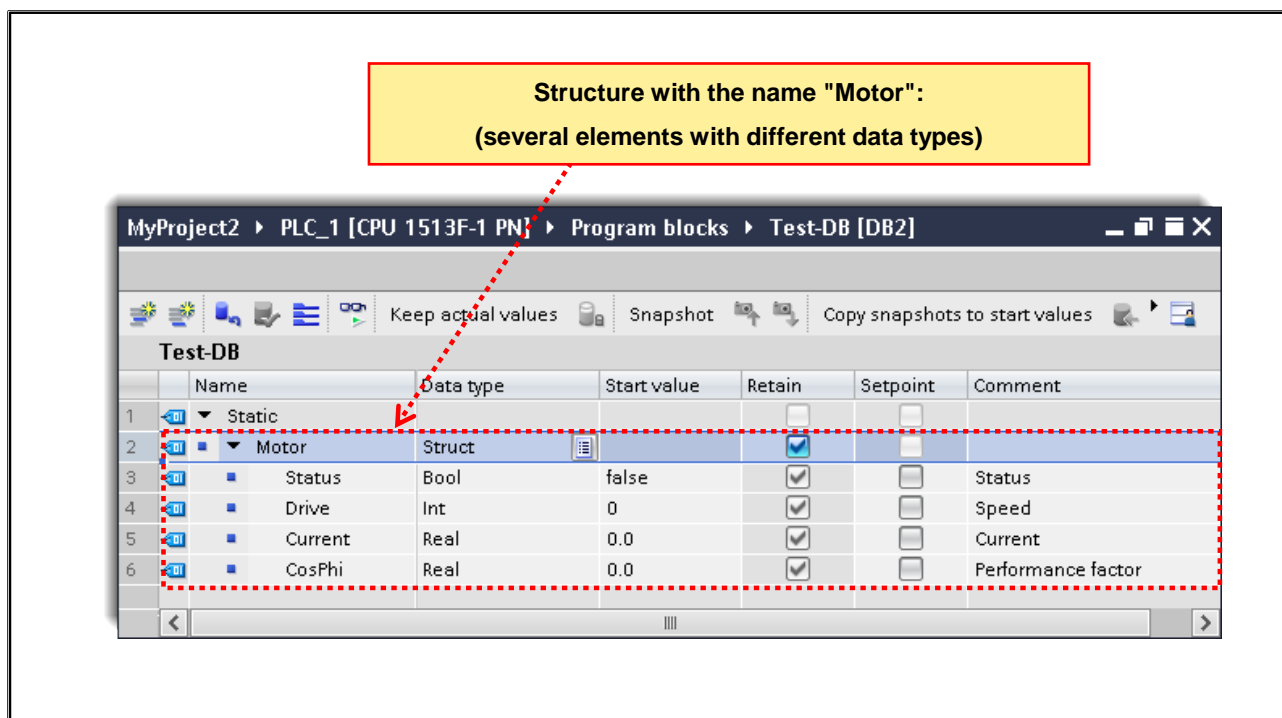
#### Array

An array consists of several elements of the same data type. In the picture above, you can see the array "Value" with 10 elements of the data type LREAL. Later, various measured values are to be stored in this array.

#### Declaring an Array in the DB

The data type of an array is called "ARRAY[n..m]". The first element (n) and the last element (m) are specified in the square brackets. In the example, [0..9] means 10 elements, whereby the first element is addressed with the index [0] and the last with the index [9]. Instead of [0..9] you could, for example, define [20..29]. This only influences the access to the elements.

### 11.5.2. Example of a Variable of the Data Type STRUCTURE



#### Structure

The picture shows an example of a structure named "Motor". The structure consists of several elements of different data types. The individual elements of a structure can be elementary or complex data types.

The access to the individual elements of a structure contains the structure name and the name of the element. This makes the program easier to read.

Example: accessing individual elements of a structure:

- "Test-DB".Motor.Status
- "Test-DB".Motor.Drive

"Test-DB" is the symbol name of the data block which contains the structure. After the symbol name, (separated by a dot) the structure name is specified. After the structure name (separated by a dot) an element name of the structure follows.

#### Declaring a Structure in the DB

As a keyword for a structure, "STRUCT" is used. The end of the structure is automatically identified with "END\_STRUCT". A name is defined for the structure (in the example: "Motor").

### 11.5.3. PLC Data Type

The screenshot illustrates the process of creating a PLC data type and its reuse in a test database. The top window, titled 'MyProject2 > PLC\_1 [CPU 1513F-1 PN] > PLC data types > Motordata', shows the 'Motordata' data type structure. The bottom window, titled 'MyProject2 > PLC > Test-DB', shows the 'Test-DB' table with columns for Name, Data type, Start value, Retain, Setpoint, and Comment. A yellow box with the text 'Structure created once can always be re-used' highlights the reuse of the 'Motordata' structure for Motor1, Motor2, and Motor3. Red dashed arrows indicate the reuse of the 'Motordata' structure for Motor1, Motor2, and Motor3.

Name	Data type	Default value	Setpoint	Comment
1 Status	Bool	false		Status
2 Drive	Int	0		Speed
3 Current	Real	0.0		Current
4 CosPhi	Real	0.0		Performance factor

Name	Data type	Start value	Retain	Setpoint	Comment
1 Static					
2 Motor1	"Motordata"				
3 Status	Bool	false			Status
4 Drive	Int	0			Speed
5 Current	Real	0.0			Current
6 CosPhi	Real	0.0			Performance factor
7 Motor2	"Motordata"				
8 Motor3	"Motordata"				

#### PLC Data Types

PLC data types are data types defined by you that are used as templates for declaring parameters and variables of complex data types (e.g. structure variables). PLC data types are created and stored in the PLC data types folder, and contain a data structure that is made up of elementary and/or complex data types. In the declaration of a variable according to PLC data type, a structure variable is created whose inner data structure is defined by the PLC data type. PLC data types can be used for the declaration of variables in global data blocks and within blocks for the declaration of local variables (TEMP, STAT) as well as parameters (IN, OUT and INOUT).



### 11.5.4. Functions RD\_SYS\_T and RD\_LOC\_T

**Data type can be set**

Test-DB				
	Name	Data type	Start value	Monitor value
5	Systemtime	DTL	DTL#1970-01-01-00:00:00	DTL#2012-07-22-06:00:58.037425293
6	YEAR	UInt	1970	2012
7	MONTH	UInt	1	7
8	DAY	UInt	1	22
9	WEEKDAY	UInt	5	1
10	HOUR	UInt	0	21
11	MINUTE	UInt	0	49
12	SECOND	UInt	0	9
13	NANOSECOND	UInt	0	404_351_879
14	Localtime	LDT	LDT#1970-01-01-00:00:00	LDT#2012-07-22-07:00:58.037464928

In the Task Card Instructions > Extended instructions > Date and time-of-day you will find functions and instructions that are intended specifically for the handling of Date / time data types.

Here you will find functions with which, for example, times can be linked, compared, written, read out etc., with one another.

#### RD\_SYS\_T

You use this instruction to read the current date and the current time-of-day (module time) of the CPU clock.

#### RD\_LOC\_T

You use this instruction to read the current local time from the CPU clock and output this at the output OUT. To display the local time, the information about time zone as well as the start of daylight saving time and standard time which you set during the configuration of the CPU clock is used.