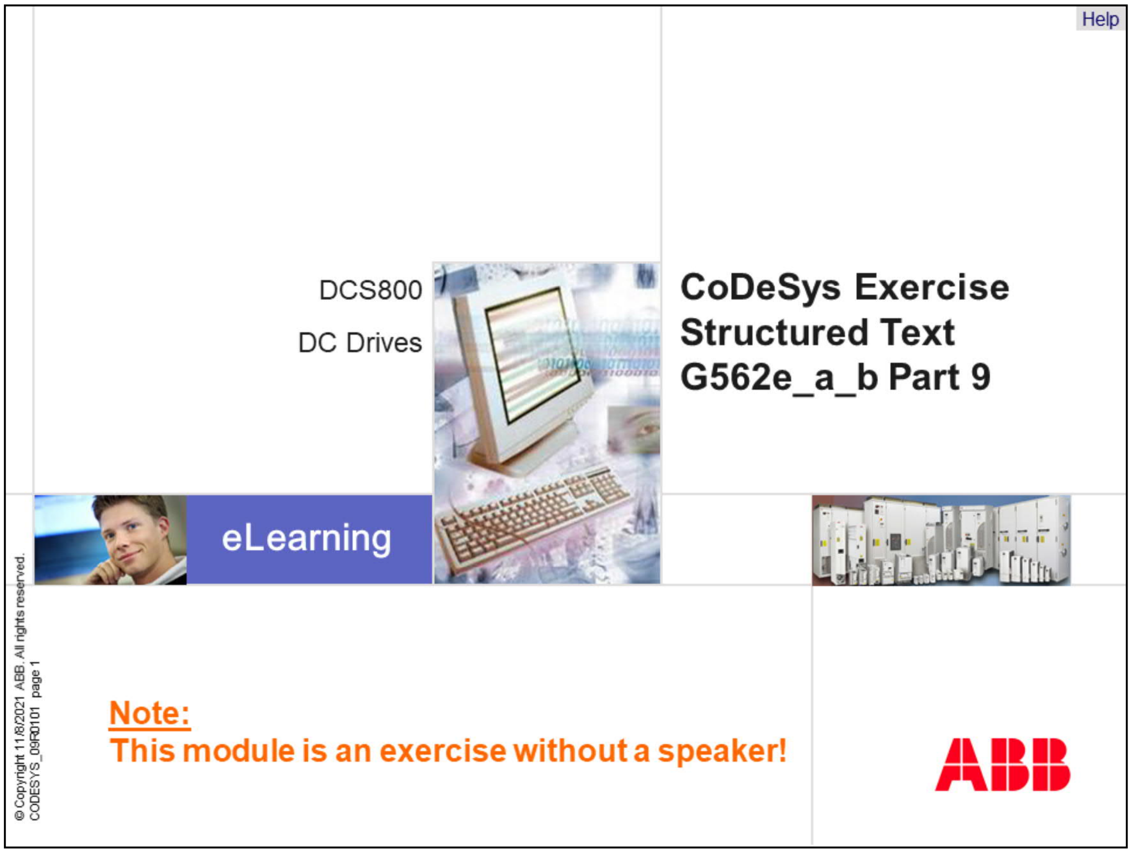


Help

DCS800

DC Drives

CoDeSys Exercise Structured Text G562e_a_b Part 9

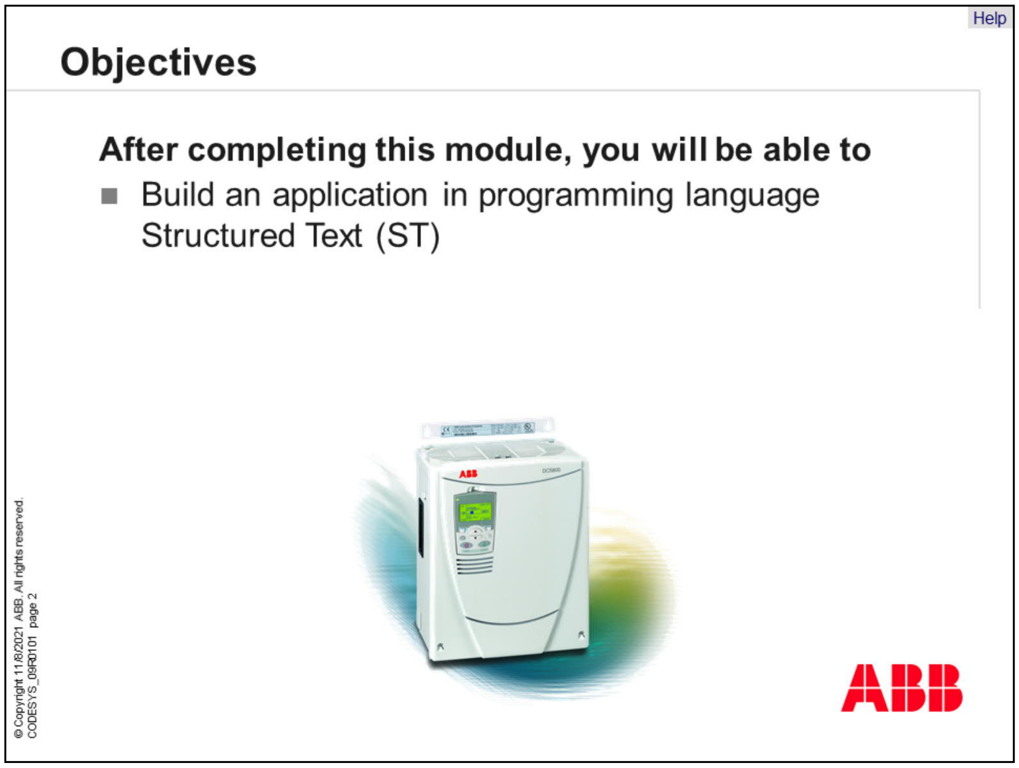
eLearning

Note:

This module is an exercise without a speaker!

Welcome to the CoDeSys training module for the DCS800, ABB DC Drives.

If you need help navigating this module, please click the Help button in the top right-hand corner. To view the presenter notes as text, please click the Notes button in the bottom right corner.



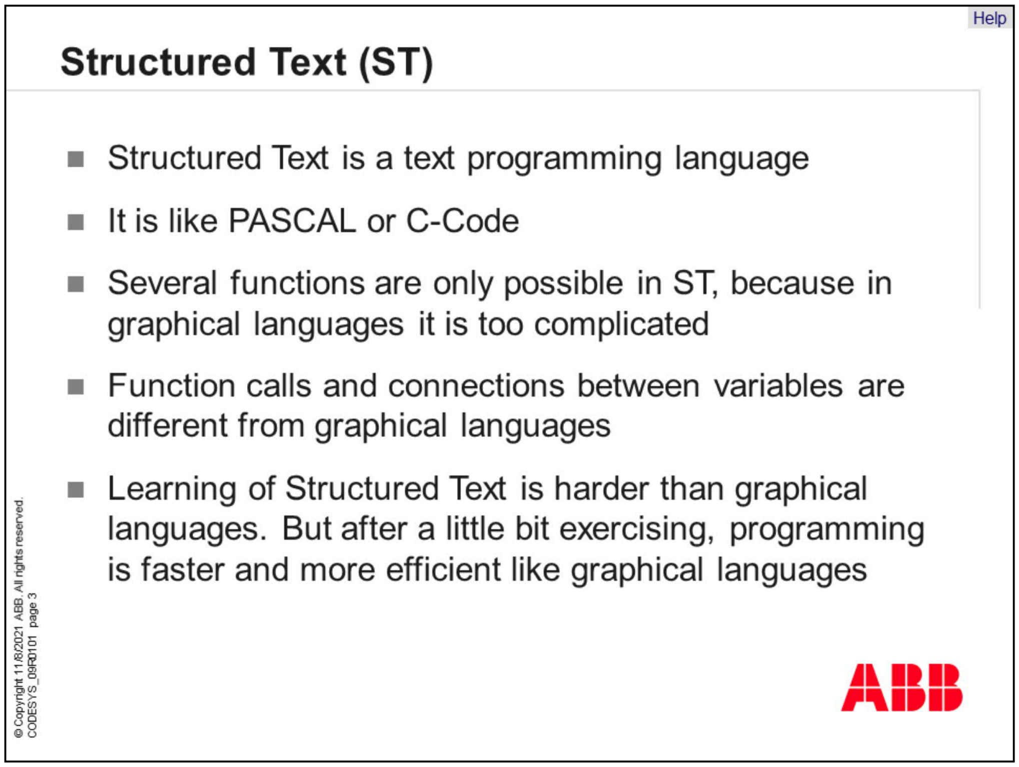
Help

Objectives

After completing this module, you will be able to

- Build an application in programming language Structured Text (ST)

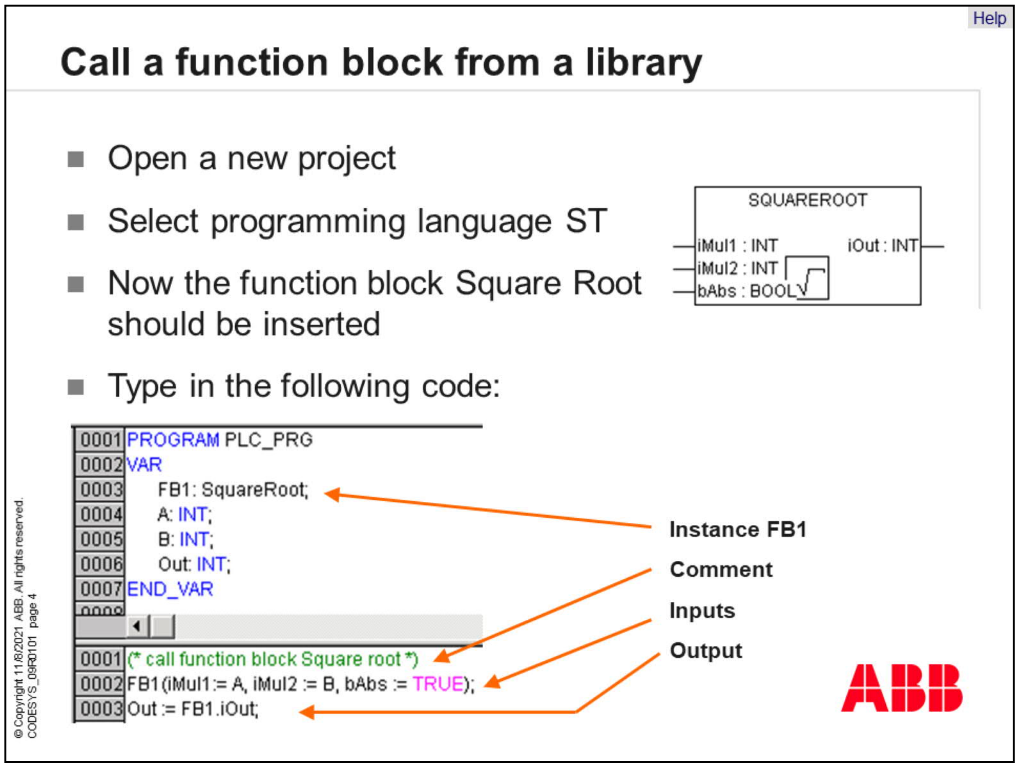
After completing of this module, you will be able to build an application using programming language "Structured Text"


Help

Structured Text (ST)

- Structured Text is a text programming language
- It is like PASCAL or C-Code
- Several functions are only possible in ST, because in graphical languages it is too complicated
- Function calls and connections between variables are different from graphical languages
- Learning of Structured Text is harder than graphical languages. But after a little bit exercising, programming is faster and more efficient like graphical languages

“Structured Text” is a text-based programming language in CoDeSys. It is similar to PASCAL or C-Code. Several functions are only possible in “Structured Text” because in graphical languages it is too complex. Function calls and connections between variables are different from graphical languages. Learning of “Structured Text” is harder than graphical languages. But after a little bit exercising, programming is faster and more efficient like graphical languages.



Help

Call a function block from a library

- Open a new project
- Select programming language ST
- Now the function block Square Root should be inserted
- Type in the following code:

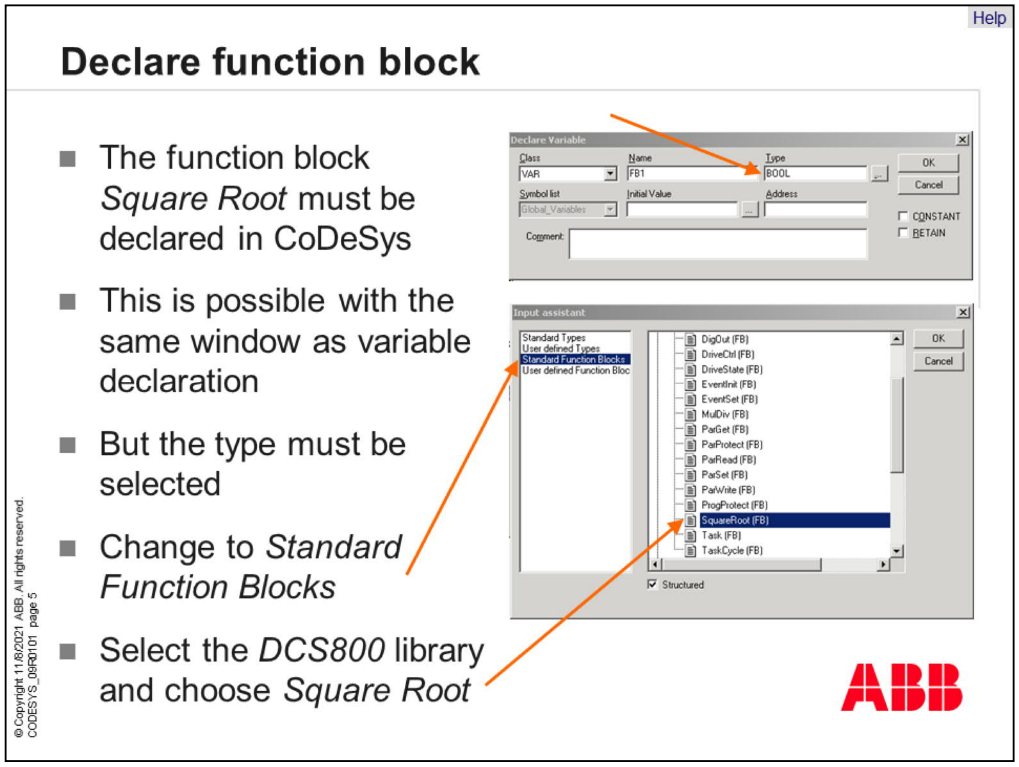
```
PROGRAM PLC_PRG
VAR
    FB1: SquareRoot;
    A: INT;
    B: INT;
    Out: INT;
END_VAR
```

```
(* call function block Square root *)
FB1(iMul1:= A, iMul2 := B, bAbs := TRUE);
Out := FB1.iOut;
```

In graphical languages a call of a function block is easy to do when you put in function blocks. In a text language the function block isn't visible at all. You see only a line with variables and allocations. But with "Structured Text" you can do the same things and more as with graphical languages.

If you will call a function block from the library type in an instance, for example "FB1". Then set brackets and click F2. Now you can select the function block from the library. What you see inside the brackets are the inputs of the selected function block. Connection between an input and other functions is possible with variables.

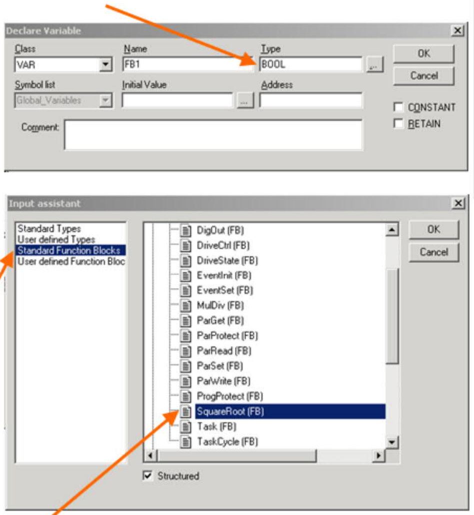
Outputs are not inside the brackets because it is easier to use the instance of the function block with the selected output.



Help

Declare function block

- The function block *Square Root* must be declared in CoDeSys
- This is possible with the same window as variable declaration
- But the type must be selected
- Change to *Standard Function Blocks*
- Select the *DCS800* library and choose *Square Root*



Declare Variable

Class	Name	Type
VAR	FB1	BOOL

Symbol list	Initial Value	Address
Global_Variables		

Comment:

☐ CONSTANT
☐ RETAIN

Input assistant

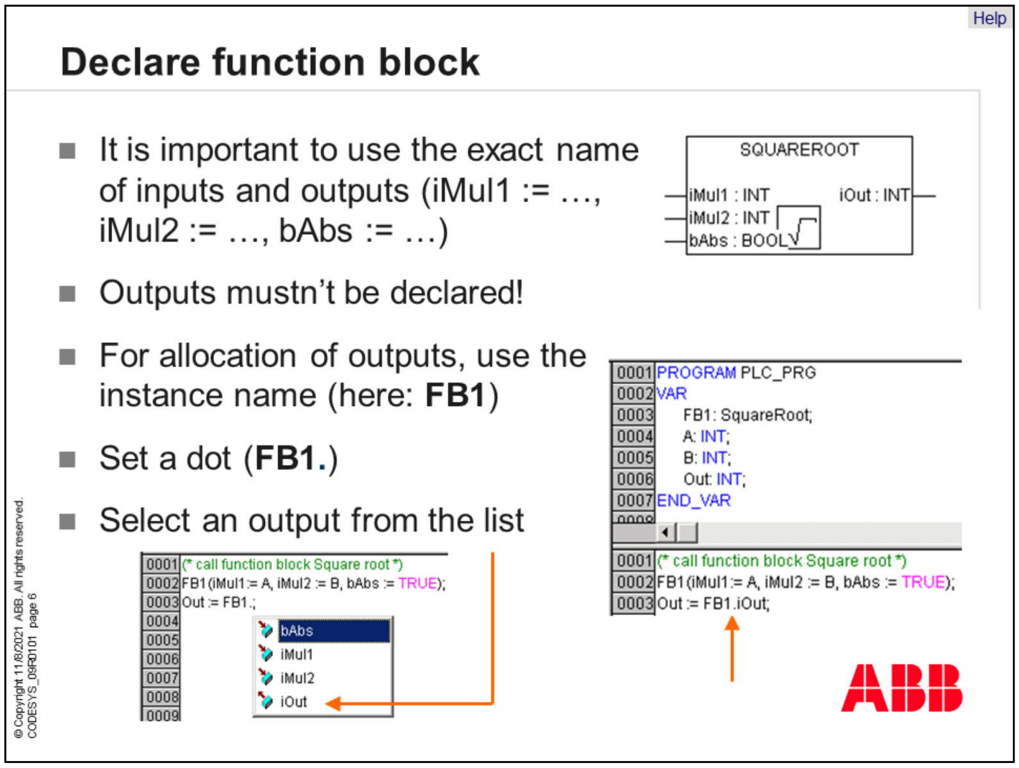
- Standard Types
- User defined Types
- Standard Function Blocks
- User defined Function Bloc

- DigOut (FB)
- DriveCtrl (FB)
- DriveState (FB)
- EventInit (FB)
- EventSet (FB)
- MulDiv (FB)
- ParGet (FB)
- ParProtect (FB)
- ParRead (FB)
- ParSet (FB)
- ParWrite (FB)
- ProgProtect (FB)
- SquareRoot (FB)
- Task (FB)
- TaskCycle (FB)

☑ Structured

ABB

Now we look deeper in the function block declaration. If a new instance is created, you can set the type. To do this use the input assistant, please. Find the several libraries which are included in item “Standard Function Blocks”. Then select the needed function block from the list. Therefore, the function block is declared.



Help

Declare function block

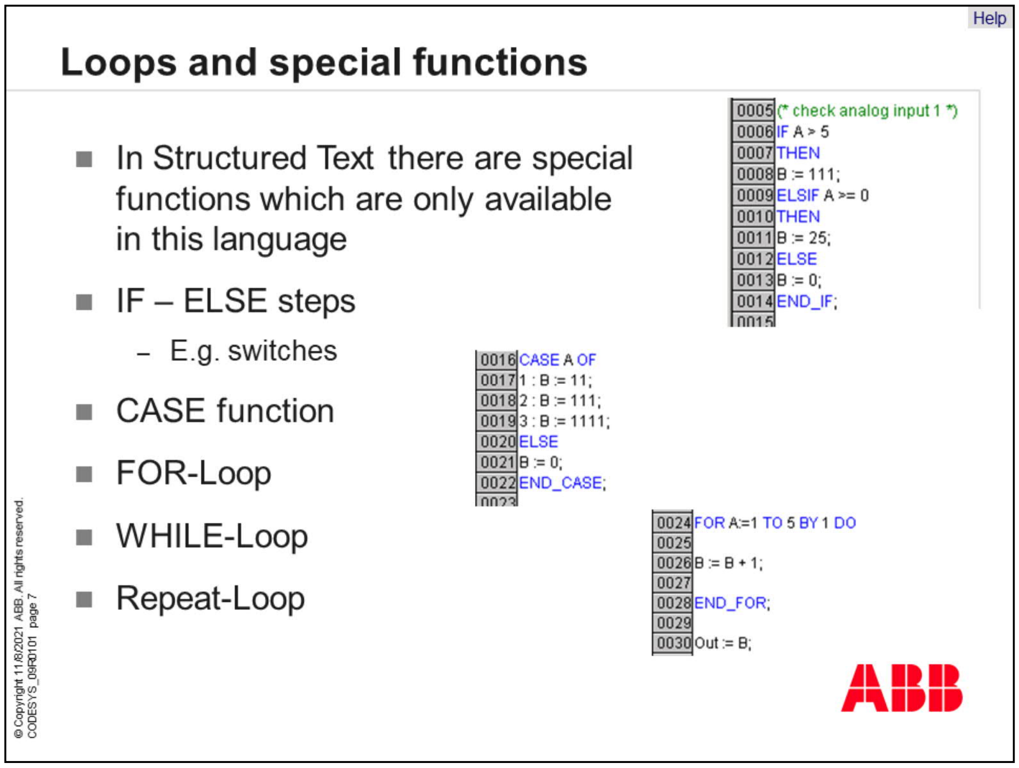
- It is important to use the exact name of inputs and outputs (iMul1 := …, iMul2 := …, bAbs := …)
- Outputs mustn't be declared!
- For allocation of outputs, use the instance name (here: **FB1**)
- Set a dot (**FB1.**)
- Select an output from the list

```
PROGRAM PLC_PRG
VAR
    FB1: SquareRoot;
    A: INT;
    B: INT;
    Out: INT;
END_VAR
```

```
(* call function block Square root *)
FB1(iMul1 := A, iMul2 := B, bAbs := TRUE);
Out := FB1.;
```

```
(* call function block Square root *)
FB1(iMul1 := A, iMul2 := B, bAbs := TRUE);
Out := FB1.iOut;
```

Inside the brackets you can see the several inputs. After the equal sign you can set the connection variable. It is important to use the exact name of inputs and outputs, otherwise you get error messages. Outputs mustn't be declared. For allocation of outputs, use the instance name, here FB1.



Help

Loops and special functions

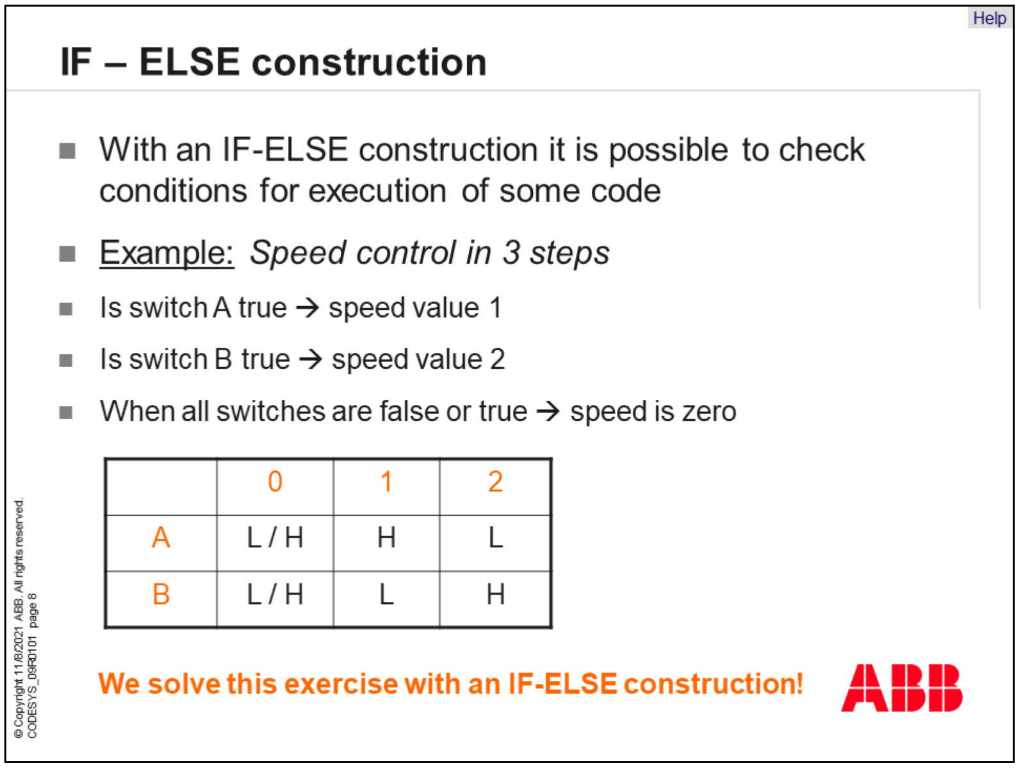
- In Structured Text there are special functions which are only available in this language
- IF – ELSE steps
  - E.g. switches
- CASE function
- FOR-Loop
- WHILE-Loop
- Repeat-Loop

```
(* check analog input 1 *)
IF A > 5
THEN
B := 111;
ELSIF A >= 0
THEN
B := 25;
ELSE
B := 0;
END_IF;
```

```
CASE A OF
1 : B := 11;
2 : B := 111;
3 : B := 1111;
ELSE
B := 0;
END_CASE;
```

```
FOR A:=1 TO 5 BY 1 DO

B := B + 1;

END_FOR;

Out := B;
```

Next special function are the loops. In “Structured Text” it is easy to define loops directly like in C-Code or Pascal. Another important function is the “IF-ELSE-Construction”. With this it’s easy to program switches. Also important is the “CASE-Command”. You can use it for selections from a volume.



Help

IF – ELSE construction

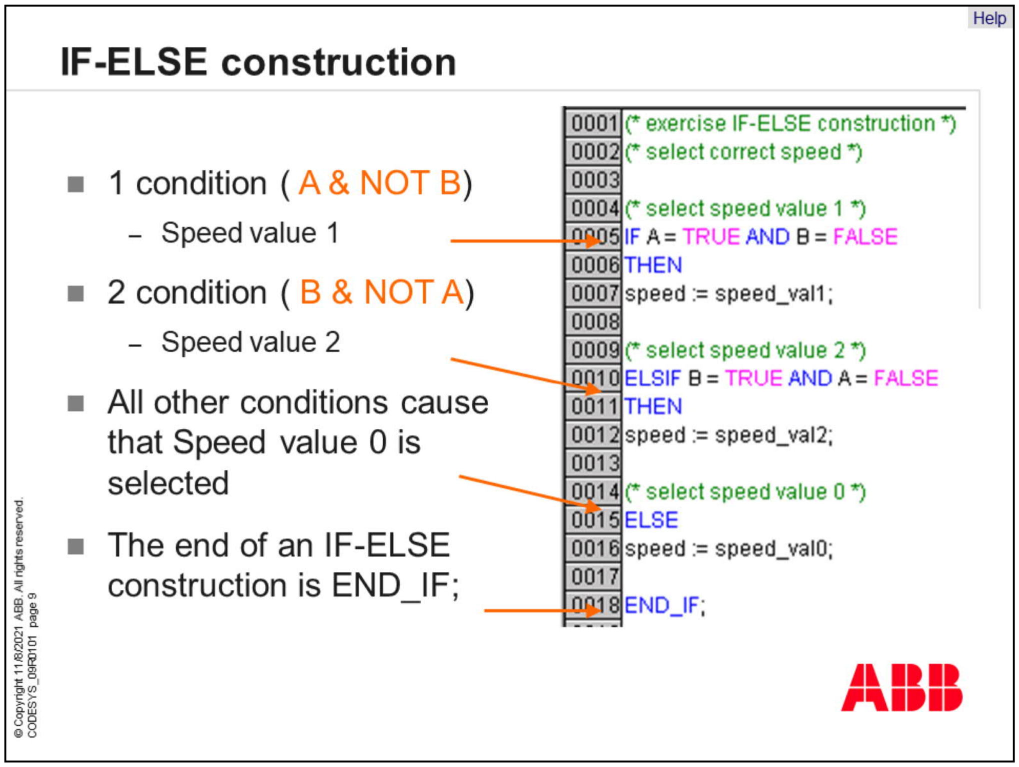
- With an IF-ELSE construction it is possible to check conditions for execution of some code
- Example: *Speed control in 3 steps*
- Is switch A true → speed value 1
- Is switch B true → speed value 2
- When all switches are false or true → speed is zero

	0	1	2
A	L / H	H	L
B	L / H	L	H

We solve this exercise with an IF-ELSE construction!

IF-ELSE construction

- With an IF-ELSE construction it is possible to check conditions for execution of some code.



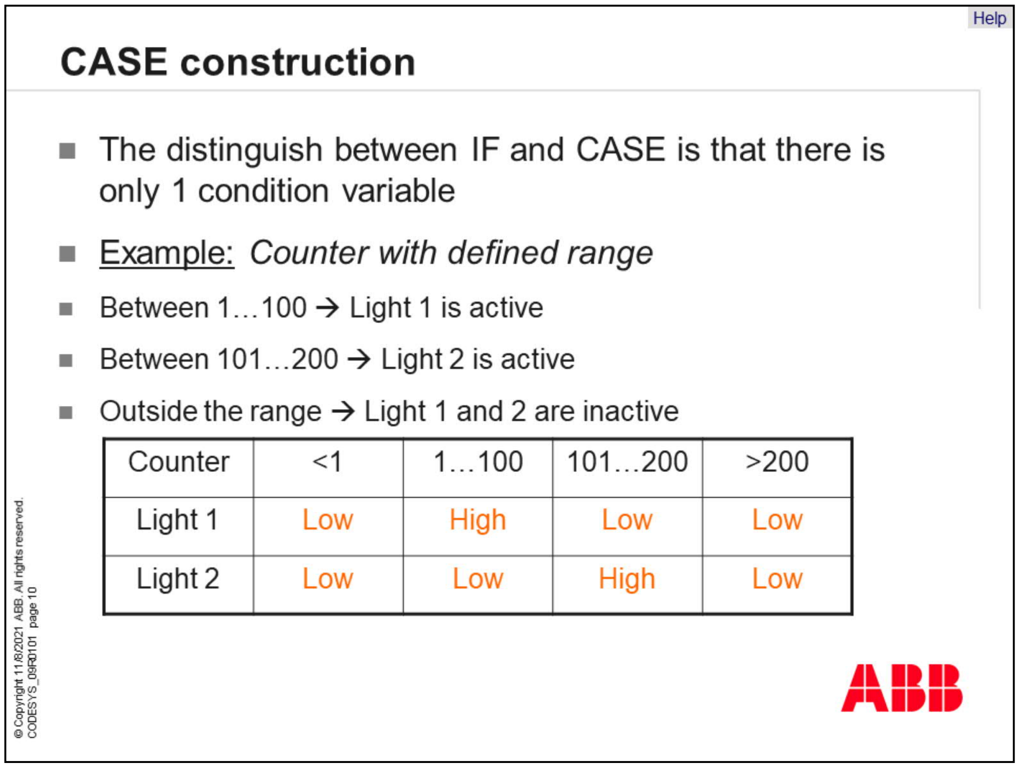
Help

IF-ELSE construction

- 1 condition (A & NOT B)
  - Speed value 1
- 2 condition (B & NOT A)
  - Speed value 2
- All other conditions cause that Speed value 0 is selected
- The end of an IF-ELSE construction is END_IF;

```
(* exercise IF-ELSE construction *)
(* select correct speed *)

(* select speed value 1 *)
IF A = TRUE AND B = FALSE
THEN
speed := speed_val1;

(* select speed value 2 *)
ELSIF B = TRUE AND A = FALSE
THEN
speed := speed_val2;

(* select speed value 0 *)
ELSE
speed := speed_val0;

END_IF;
```

Let’s have a look to the “IF-ELSE-Construction”. Each “IF-Condition” has a boolean expression to select if the expression is fulfilled. If it’s fulfilled the command will be executed. Otherwise, the next step is checked. The whole construction ends with the command “END_IF”. So, you can define a stairs with several steps which can be executed or not.



Help

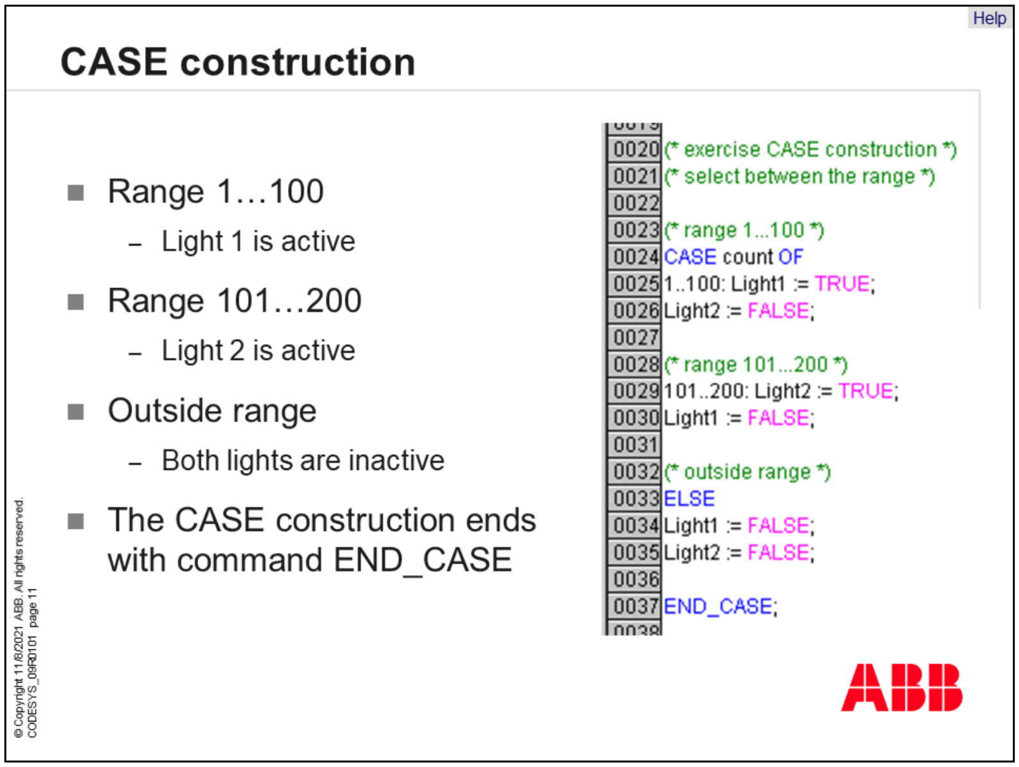
CASE construction

- The distinguish between IF and CASE is that there is only 1 condition variable
- Example: *Counter with defined range*
- Between 1…100 → Light 1 is active
- Between 101…200 → Light 2 is active
- Outside the range → Light 1 and 2 are inactive

Counter	<1	1…100	101…200	>200
Light 1	Low	High	Low	Low
Light 2	Low	Low	High	Low

Now we change to the “CASE-Construction”. The distinguish between “IF” and “CASE” is that there is only 1 condition variable. This variable has more than 2 conditions and the “CASE-Construction” checks the actual state of the variable. Then it comes a decision!

An example is a counter with a defined range. The selection works between the defined ranges. So, the counter goes up and down and the result is shown at light 1 or 2 if the counter is inside the defined range.



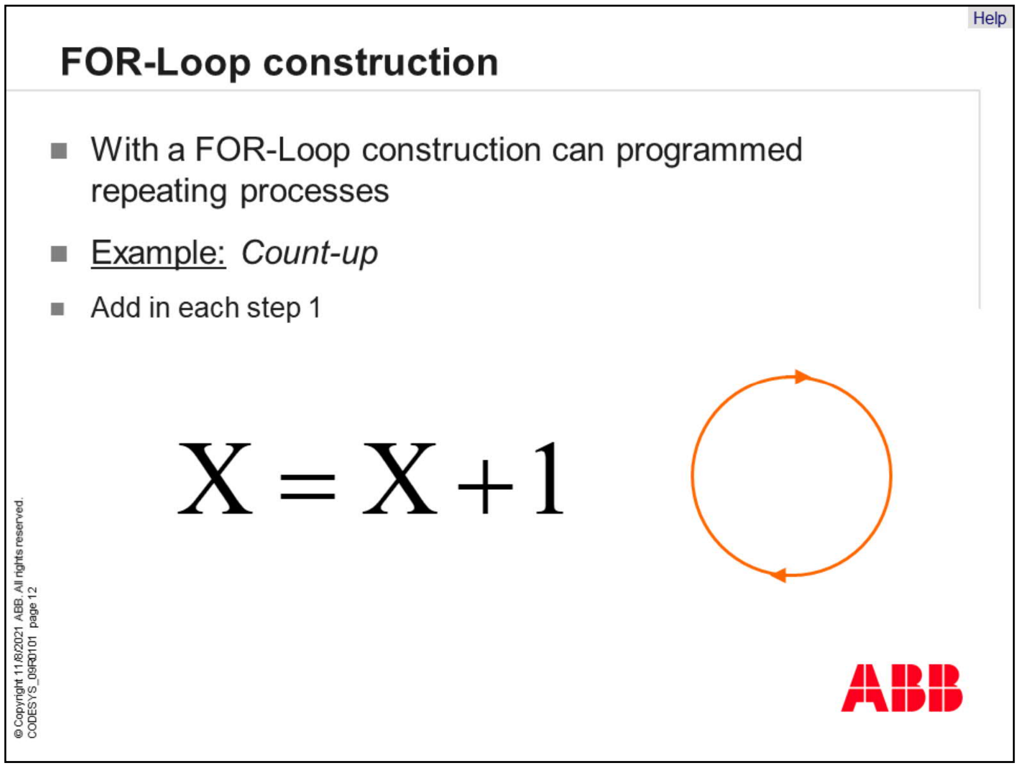
Help

CASE construction

- Range 1…100
  - Light 1 is active
- Range 101…200
  - Light 2 is active
- Outside range
  - Both lights are inactive
- The CASE construction ends with command END_CASE

```
(* exercise CASE construction *)
(* select between the range *)

(* range 1...100 *)
CASE count OF
1..100: Light1 := TRUE;
Light2 := FALSE;

(* range 101...200 *)
101..200: Light2 := TRUE;
Light1 := FALSE;

(* outside range *)
ELSE
Light1 := FALSE;
Light2 := FALSE;

END_CASE;
```

The “CASE-Construction” looks like the picture on the right side. We have 1 variable, here “count”, and the several steps with the defined range. It ends with the command “END_CASE”.



Help

FOR-Loop construction

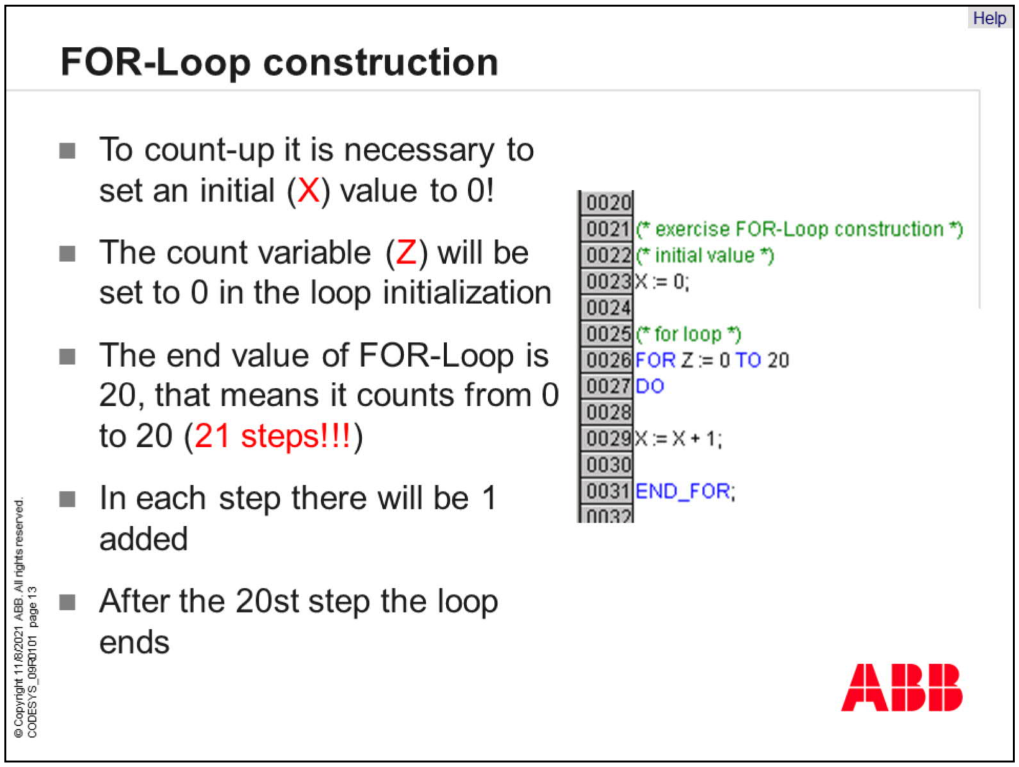
- With a FOR-Loop construction can programmed repeating processes
- Example: *Count-up*
- Add in each step 1

$$X = X + 1$$

© Copyright 11/8/2021 ABB. All rights reserved.
CODESYS_09R0101 page 12

ABB

Another important function is the "FOR-Loop-Construction". It have the functionality like a counter because the loop will be executed as long as the condition is fulfilled. An example is an upward counter. The user defined the starting point and the end point. Later the loop runs from the starting point to the end and increase the counter value with each cycle.



Help

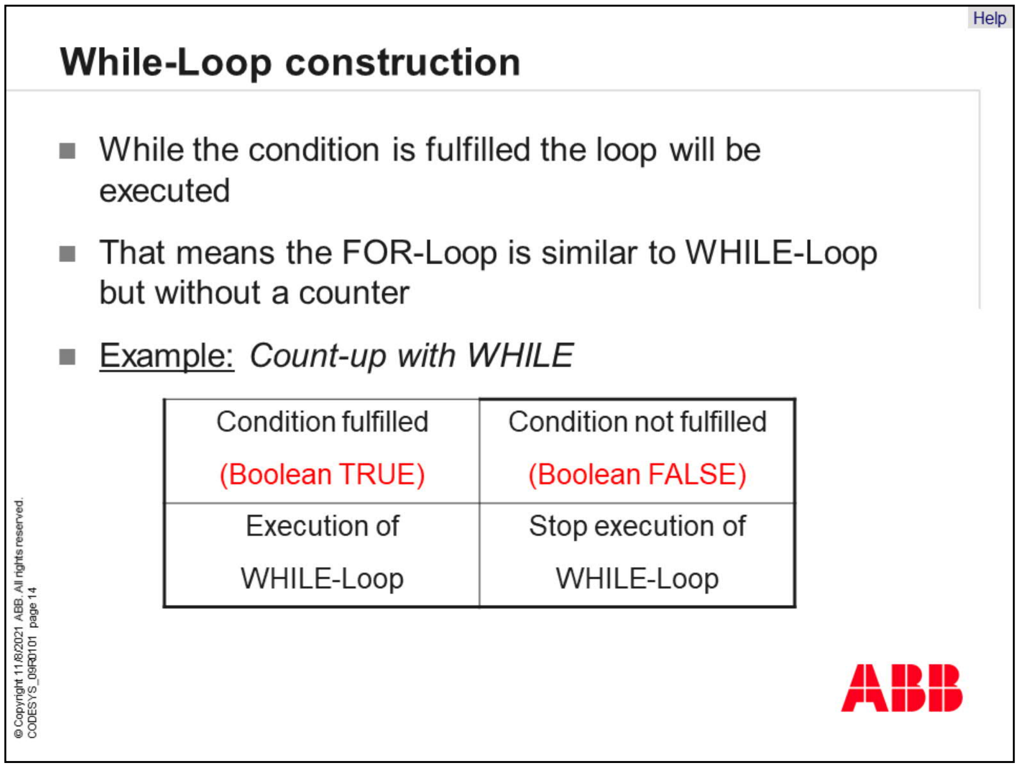
FOR-Loop construction

- To count-up it is necessary to set an initial (X) value to 0!
- The count variable (Z) will be set to 0 in the loop initialization
- The end value of FOR-Loop is 20, that means it counts from 0 to 20 (21 steps!!!)
- In each step there will be 1 added
- After the 20st step the loop ends

```
(* exercise FOR-Loop construction *)
(* initial value *)
X := 0;

(* for loop *)
FOR Z := 0 TO 20
DO

X := X + 1;

END_FOR;
```

FOR loop construction

- The picture shows the FOR-loop construction.
- It begins with FOR, the initialization value and the limit.
- The command in the loop starts after the DO command.


Help

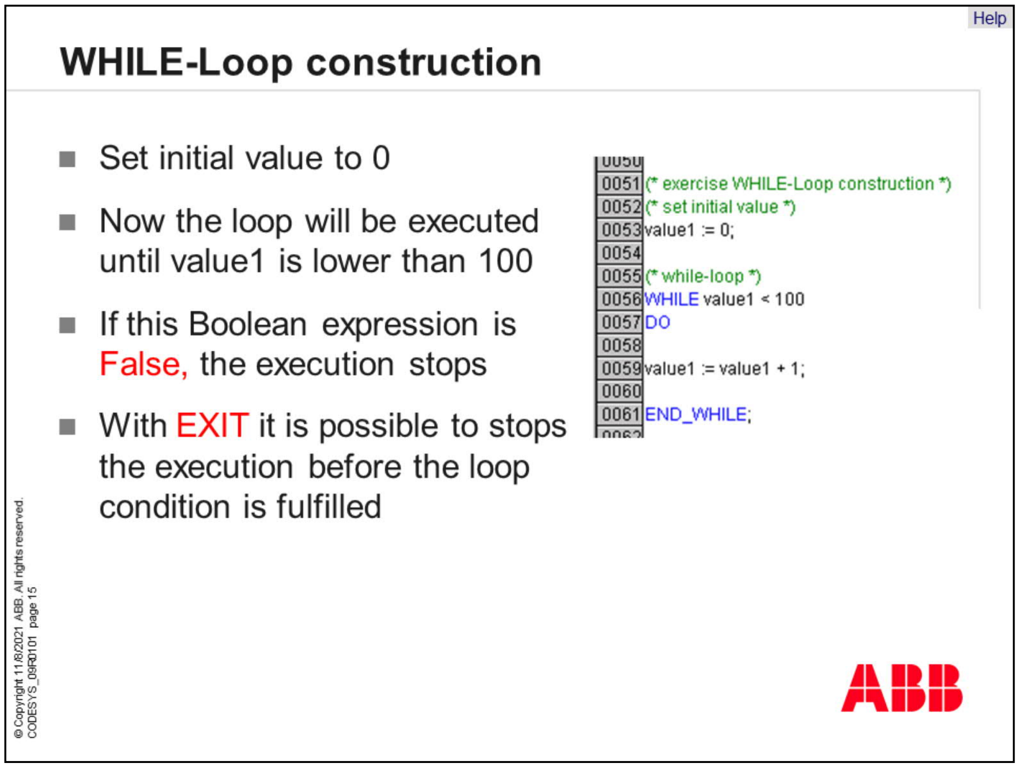
While-Loop construction

- While the condition is fulfilled the loop will be executed
- That means the FOR-Loop is similar to WHILE-Loop but without a counter
- Example: *Count-up with WHILE*

Condition fulfilled (Boolean TRUE)	Condition not fulfilled (Boolean FALSE)
Execution of WHILE-Loop	Stop execution of WHILE-Loop

WHILE-Loop construction

- While the condition is fulfilled the loop will be executed.
- That means the FOR-Loop is like the WHILE-Loop but without a counter



Help

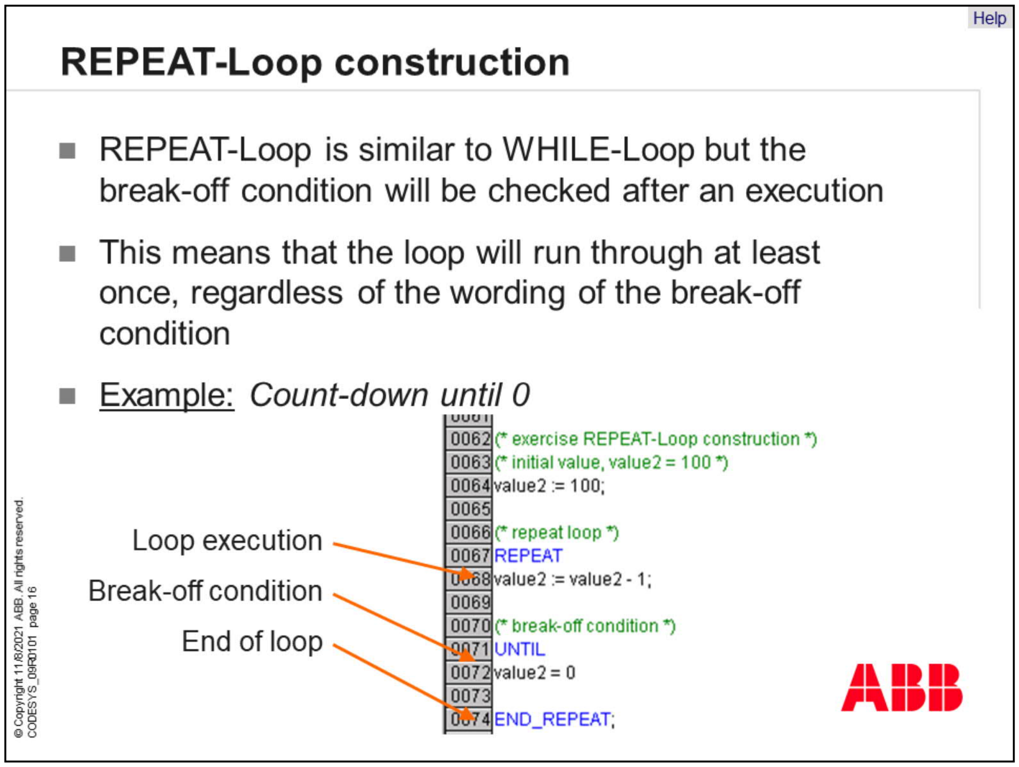
WHILE-Loop construction

- Set initial value to 0
- Now the loop will be executed until value1 is lower than 100
- If this Boolean expression is False, the execution stops
- With EXIT it is possible to stops the execution before the loop condition is fulfilled

```
(* exercise WHILE-Loop construction *)
(* set initial value *)
value1 := 0;

(* while-loop *)
WHILE value1 < 100
DO

value1 := value1 + 1;

END_WHILE;
```

WHILE-Loop construction

- An example for a WHILE-Loop shows the picture.



Help

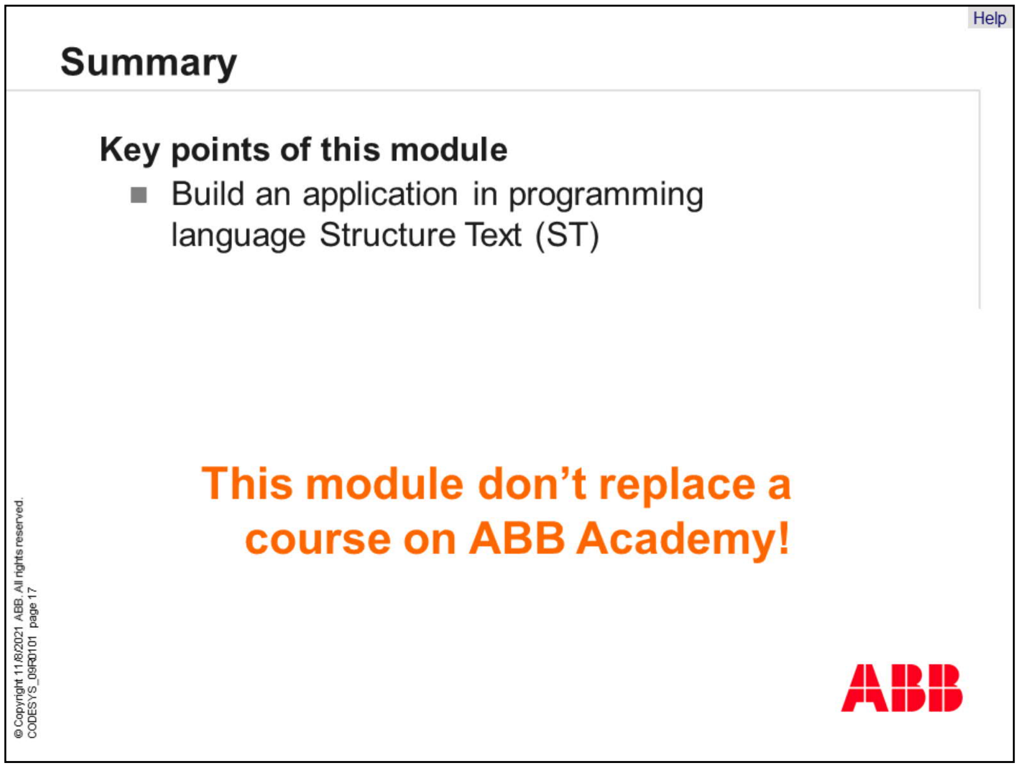
REPEAT-Loop construction

- REPEAT-Loop is similar to WHILE-Loop but the break-off condition will be checked after an execution
- This means that the loop will run through at least once, regardless of the wording of the break-off condition
- Example: *Count-down until 0*

```
(* exercise REPEAT-Loop construction *)
(* initial value, value2 = 100 *)
value2 := 100;

(* repeat loop *)
REPEAT
value2 := value2 - 1;

(* break-off condition *)
UNTIL
value2 = 0

END_REPEAT;
```

REPEAT loop construction

- The REPEAT-Loop is like the WHILE-Loop, but the break-off condition will be checked after an execution.
- This means that the loop will run through at least once, regardless of the wording of the break-off condition.



Help

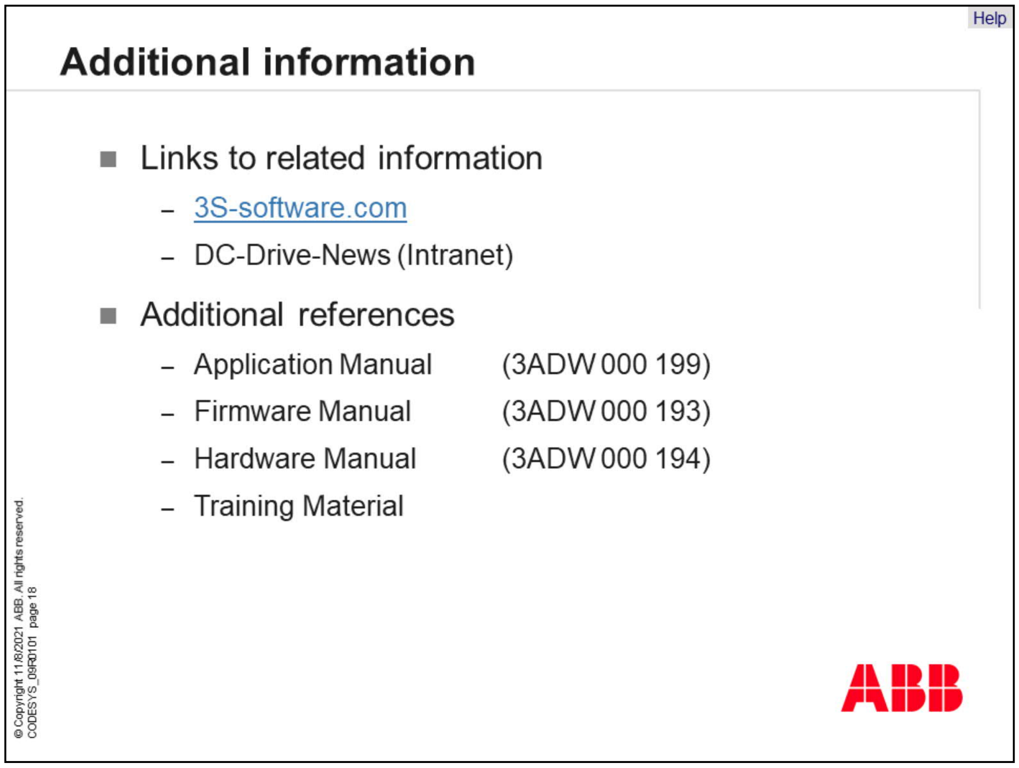
Summary

Key points of this module

- Build an application in programming language Structure Text (ST)

This module don’t replace a course on ABB Academy!

- If you will learn more about CoDeSys application programming, use a course on ABB Academy, please.

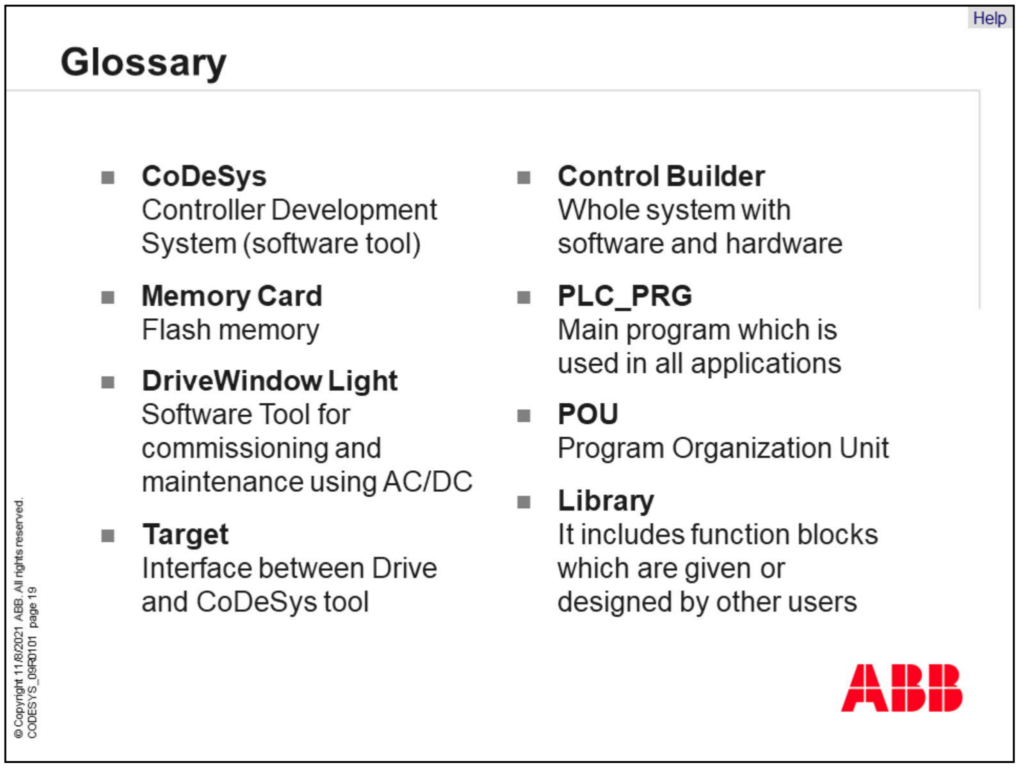


Help

Additional information

- Links to related information
  - 3S-software.com
  - DC-Drive-News (Intranet)
- Additional references

– Application Manual	(3ADW 000 199)
– Firmware Manual	(3ADW 000 193)
– Hardware Manual	(3ADW 000 194)
– Training Material	



Help

Glossary

- **CoDeSys**
  Controller Development System (software tool)
- **Memory Card**
  Flash memory
- **DriveWindow Light**
  Software Tool for commissioning and maintenance using AC/DC
- **Target**
  Interface between Drive and CoDeSys tool
- **Control Builder**
  Whole system with software and hardware
- **PLC_PRG**
  Main program which is used in all applications
- **POU**
  Program Organization Unit
- **Library**
  It includes function blocks which are given or designed by other users

Power and productivity for a better world™

Thank you for your attention. You may now go ahead and move on to the next unit.