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.
After completing of this module, you will be able to build an application using programming language "Structured Text"
“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.
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.
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.
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.
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.
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 $\Rightarrow$ speed value 1
  - Is switch B true $\Rightarrow$ speed value 2
  - When all switches are false or true $\Rightarrow$ speed is zero

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>L/H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>B</td>
<td>L/H</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>

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.
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.
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.

**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

<table>
<thead>
<tr>
<th>Counter</th>
<th>&lt;1</th>
<th>1…100</th>
<th>101…200</th>
<th>&gt;200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light 1</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Light 2</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
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”. 
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.
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.
**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*

<table>
<thead>
<tr>
<th>Condition fulfilled (Boolean TRUE)</th>
<th>Condition not fulfilled (Boolean FALSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution of WHILE-Loop</td>
<td>Stop execution of WHILE-Loop</td>
</tr>
</tbody>
</table>

**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.
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 stop the execution before the loop condition is fulfilled

WHILE-Loop construction
- An example for a WHILE-Loop shows the picture.
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.
If you will learn more about CoDeSys application programming, use a course on ABB Academy, please.
Additional information

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

- Additional references
  - Application Manual (3ADW000 199)
  - Firmware Manual (3ADW000 193)
  - Hardware Manual (3ADW000 194)
  - Training Material
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
Thank you for your attention. You may now go ahead and move on to the next unit.