# Festo Software Tools

FESTO

**Textbook**

Automating
with FST

682 300
en 0402NH

Author ..... B. Plagemann

Original ..... de

Edition ..... en 0402NH

Designation ..... P.BE-FST4-LB-EN

Order no. ..... 682 300


Internet: http://www.festo.com
e-mail: service_international@festo.com

Microsoft® Windows® registered trade mark of Microsoft Corporation

# Contents

## Designated use

This software package enables any user familiar with PLC/IPC drives to configure, program and commission the SPS/IPC supported by the software package.

Observe also the standards specified in the relevant chapters, as well as national and local laws and technical regulations.

## Target group

This manual is intended for technicians trained in control and automation technology, fitting, etc. who are dealing for the first time with commissioning and operating the PLC/IPC supported by the software.

Information:
You will find software complementary to the product on the Festo Internet pages under the address: www.festo.com under [Industrial Automation] [Services & Support] [Download Area].

## Example projects

The sample projects from this book are archived on the FST CD in the subdirectory \Samples\Deutsch or \Samples\English. If necessary, you can read them using FST (restore).

If you also installed sample files when you installed FST, the example projects are already located in the FST project directory.

## Important user instructions

### Danger categories

This manual contains instructions on the possible dangers which may occur if the product is not used correctly. These instructions are marked (Warning, Caution, etc.), printed on a shaded background and marked additionally with a pictogram. A distinction is made between the following danger warnings:

**Warning**
This means that failure to observe this instruction may result in serious personal injury or damage to property.

**Caution**
This means that failure to observe this instruction may result in personal injury or damage to property.

**Please note**
This means that failure to observe this instruction may result in damage to property.

The following pictogram marks passages in the text which describe activities with electrostatically sensitive components.

Electrostatically sensitive components may be damaged if they are not handled correctly.

## Marking special information

The following pictograms mark passages in the text containing special information.

**Pictograms**

Information:
Recommendations, tips and references to other sources of information.

Accessories:
Information on necessary or sensible accessories for the Festo product.

Environment:
Information on environment-friendly use of Festo products.

**Text markings**

- The bullet indicates activities which may be carried out in any order.

1. Figures denote activities which must be carried out in the numerical order specified.

– Hyphens indicate general activities.

# Programming for automation technology

## Chapter 1

# Contents

Automation can be defined as “using processors to automatically carry out technical processes”.
The starting point for automation is a technical process which should be carried out in such a way that certain objectives (e.g. efficiency, safety, availability) are achieved.
In order for this to take place automatically, i.e. without constant human intervention, automation equipment is required. It implements the functions for registering, processing and output of signals.

Implementation of the processing algorithm takes places as logical, arithmetic links. In the past this was based on various types of auxiliary power (e.g. electric, pneumatic, hydraulic); today this takes place almost exclusively as programs in computers.[1)]

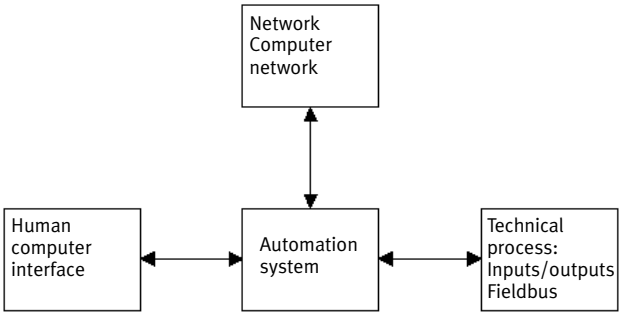



Fig. 1/1

1) http://alex.fh-trier.de/info/automatisierung.html

## 1.1 About this book

This book describes how this processing algorithm is developed, tested and optmised using FST[2)]. The objective is to enable the user to learn and experience on his or her own the basic, necessary features of the FST software with the aid of the book. All the examples have been developed with Version 4.1 FST software.

## 1.2 Programming for automation technology or the PC

Contrary to programs written for PCs or handheld computers, for example, programs for automation technology must normally function “without constant human intervention”. For instance, while the word processor into which this text is being entered requires the human user to constantly be in attendance, the welding robot must weld the automobile body without humans constantly having to intervene. However, humans are naturally needed for the “technical process” to come about in the first place. There is thus a need for the planning and programming hand as well as user intervention.

Today planning and programming is done practically all the time on PCs using programming software developed just for this purpose. Programs are then executed in the controller.

This represents a significant difference between the programming for typical PC applications and for typical automation applications.

In automation technology, the programming computers and the controller computers are two distinct, independent units.

[2)] FST Festo Software Tools

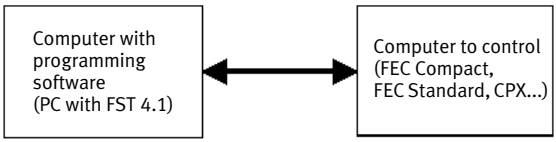



Fig. 1/2

In automation technology we thus always have to assume that at least two machines are necessary:
The programming machine and the controller machine, or simply: the controller.

# The basic rules of an FST project

## Chapter 2

# Contents

## 2.1 Overview

The following should explain programming procedures for FST in detail. You will obtain as complete an overview as possible. Everything described has a common basis for the processing of an automation projection with some basic tasks which are always necessary:

| No. | What to do | Comment |
|---|---|---|
| 1 | Call up a new project | In the Project menu |
| 2 | Name the project | In the dialog box |
| 3 | Select the proper controller, enter project comment | You can use FST software to program completely different CPUs of an entire controller family. |
| 4 | I/O configuration | There will hardly ever be an automation project without any inputs or outputs. |
| 5 | Programming | At least a Program 0 is mandatory<br>Decision whether to use a ladder diagram or a statement list |
| 6 | Make project | You will be automatically asked to do this |
| 7 | Download project | A connection to the controller must exist to do this |
| 8 | Check | And possibly correct, improve, optimise |
| 9 | Document | Describe, make comments, print |

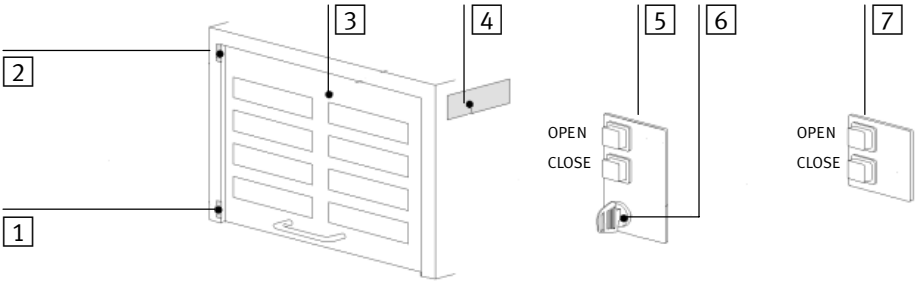
Tab. 2/1

# The first FST project: Controlling a garage door

## Chapter 3

# Contents

The first practical example should follow the example of a compact controller – and also reduced to the most simple basic functions: A garage door should open and close from the inside and outside.




1 Limit switch (lower)

2 Limit switch (upper)

3 Garage door

4 Motor up and down

5 Outside area of garage door controller

6 Key switch

7 Inside area of garage door controller

Fig. 3/1

The garage door should be controlled in such a way that

- the door can be closed at any time from the inside and outside
- the door can only be opened from the outside when the key switch and the OPEN button are actuated simultaneously
- the door can be opened from the inside at any time
- the door always runs up to the limit switch (above) when opening
- the door only closes when the CLOSE button is being pressed.

These can be represented in the following way:

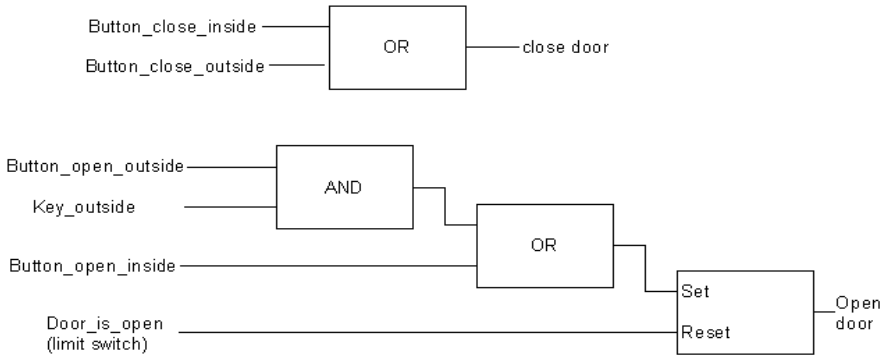



Fig. 3/2

An IPC FEC FC20 [3] is used as a controller, that is, a compact controller with 12 inputs and 8 output.

[3] Of course, all other FST controllers can also be used in this example. An IPC FEC Standard, such as the FC 400, would also work well in this example. However, an IPC PS1, which offers more upgrade possibilities due to its modular design, could also been used.

## 3.1 Project, I/Os, program, IF ... THEN ... OTHRW

We will be strictly adhering to the order in Chapter 2 [4]:

### 3.1.1 Project

- Please create a new project.


Fig. 3/3

- Now name your project 'Garage'


Fig. 3/4

[4] You will find this example as Gar_en_s.ZIP on your CD.

### 3.1.2 Selecting the controller type

- Please select the proper controller – here the FEC Compact [5] – and write a comment if you wish.

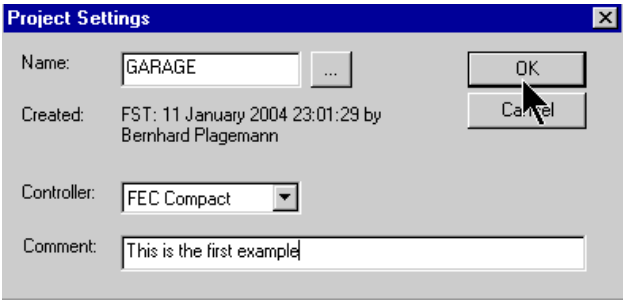



Fig. 3/5

[5] If you are using a FEC Standard, this image must look like this:


Fig. 3/6

### 3.1.3 I/O configuration

- Now open the I/O configuration by double-clicking the entry in the project window.


Fig. 3/7

You see that an I/O module has already been entered here. This is automatically entered for all controllers for which it is mandatory that a certain I/O module be available. In the column IW you will recognise that the inputs begin with I0.0, and that the outputs begin with O0.0 in the column OW. You can change this specification if necessary.

#### I/O configuration for FEC Standard

Use a controller type for which you have more possibilities for the input/output modules; you will have to use a I/O module here.

1. To do so, right-click into the empty I/O configuration and choose Insert from the context menu or double-click into the empty configuration.

2. Now open the menu and select the I/O card that you want for your application.

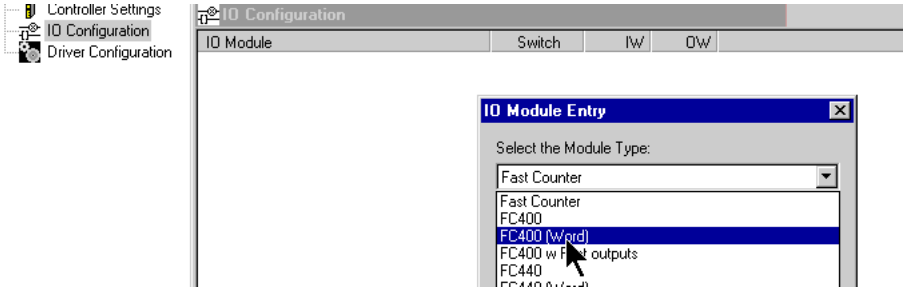



Fig. 3/8

A FEC FC400 has 16 inputs and 8 outputs. The inputs are grouped in 2 groups of one byte each (8 bits, 8 inputs) (FC400 seen from above).

Select FC400 from the menu and an input word will be assigned to each byte. The following will therefore be allocated automatically:

I0.0 ... I0.7
I1.0 ... I1.7
(O.00) O0.7

Select FC400 (Word) in the menu and two input bytes will be summarised in one word. The following will therefore be allocated automatically:

I0.0 ... I0.15
O0.0 ... O0.7

## I/O addresses

Perhaps it is not quite that easy to understand what these addresses mean.
The garage door example shows that sensors, actuators and the controller are necessary for a process. Sensors and actuators are independent devices (a limit switch, a switch, a temperature monitor, a valve, a contact, a motor ...). So that the controller can make use of these devices, there must a connection between the sensors/actuators and the controller:

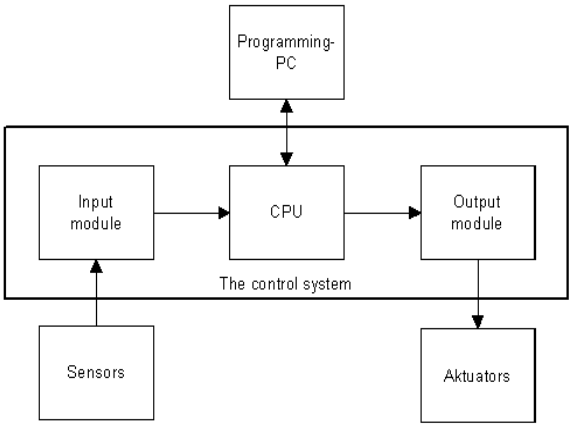



Fig. 3/9

The connection from the sensors and to the actuators takes place via the I/O modules. These I/O modules require a name so that the CPU can communicate with them. For FST naming always takes place by words, thus 'IW' for 'input word' and 'OW' for 'output word'. A single bit can then be accessed within a 'word'.
For larger or nested controllers, there can be many I/O modules.

This is basically nothing more than an 'address'. If a postman carries letters, then he has, for example, 16 different letters for 'King Street' – the controller would say: Data for the output word OW66.
The letters are distributed to 16 letterboxes on King Street – the controller would distribute to the output bits as follows: O66.0, O66.1, O66.2 .... O66.15

The allocation list for this example would look like this:

Allocation List

| Operand | Symbol | Comment |
|---|---|---|
| O66.0 | Door_0 | Mail for 0, main road |
| O66.1 | Door_1 | Mail for 1, main road |
| O66.2 | Door_2 | Mail for 2, main road |
| O66.3 | Door_3 | Mail for 3, main road |
| O66.4 | Door_4 | Mail for 4, main road |
| O66.5 | Door_5 | Mail for 5, main road |
| O66.6 | Door_6 | Mail for 6, main road |
| O66.7 | Door_7 | Mail for 7, main road |
| O66.8 | Door_8 | Mail for 8, main road |
| O66.9 | Door_9 | Mail for 9, main road |
| O66.10 | Door_10 | Mail for 10, main road |
| O66.11 | Door_11 | Mail for 11, main road |
| O66.12 | Door_12 | Mail for 12, main road |
| O66.13 | Door_13 | Mail for 13, main road |
| O66.14 | Door_14 | Mail for 14, main road |
| O66.15 | Door_15 | Mail for 15, main road |

Fig. 3/10

## 3.1.4 Programming

In practice programming is mostly started by entering the inputs and outputs into the allocation list.
We should proceed in the same manner.

### Allocation list

- Open the allocation list with a double click.

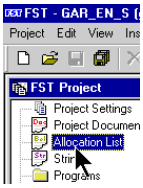



Fig. 3/11

- In the empty allocation list you can enter the first operand either again by double-clicking or by right-clicking to get the context menu.

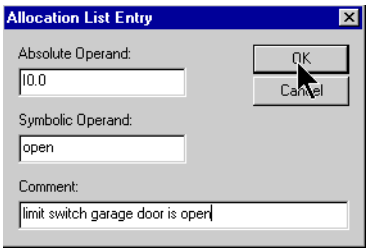
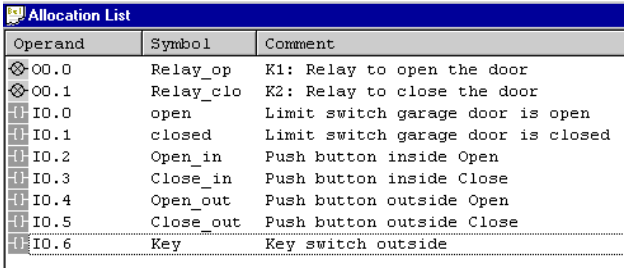



Fig. 3/12

- Now enter all the operands in the allocation list.
  The following image shows a possible allocation list.



Allocation List

| Operand | Symbol | Comment |
|---|---|---|
| O0.0 | Relay_op | K1: Relay to open the door |
| O0.1 | Relay_clo | K2: Relay to close the door |
| I0.0 | open | Limit switch garage door is open |
| I0.1 | closed | Limit switch garage door is closed |
| I0.2 | Open_in | Push button inside Open |
| I0.3 | Close_in | Push button inside Close |
| I0.4 | Open_out | Push button outside Open |
| I0.5 | Close_out | Push button outside Close |
| I0.6 | Key | Key switch outside |

Fig. 3/13

And here are some rules for the allocation list:

– The operand's address must match the entry in the I/O configuration. If IW20 were entered in the I/O configuration as an input word, then the address I20.0 and so forth would have to be used here.

– The symbol may have up to 9 characters, no special characters and also no predefined characters. For instance, I0 and V10 are prohibited as symbolic operands [6], as I0 is used for input and V10 for constant.

– The comment may consist of any characters you like.

[6] You will find a list of all FST operands in section 18.1.

## Programm a statement list [7)]

Once the allocation list has been created, it is finally time to write the program. Every FST project required a program with the No. 0.
Right-clicking on the Programs directory in the project window opens the context menu with the possibility of inserting a new program. Or use the icon for 'New Program' or the Insert menu.


Fig. 3/14

Now choose the programming language you want. Please note that you cannot automatically translate between the statement list and the ladder diagram. Specifying one of the two languages thus applies to the entire program. On the other hand, programs with various programming languages can be mixed within a project – Program 0 could therefore be written in statement list (STL), while Program 12 is in ladder diagram (LDR).

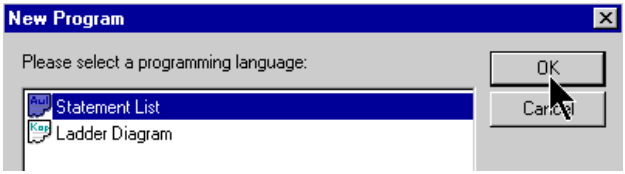



Fig. 3/15

[7)] If you would like to program in ladder diagram, please skip to the section 'Program as a ladder diagram' on page 3-20.

The following entries are now possible:

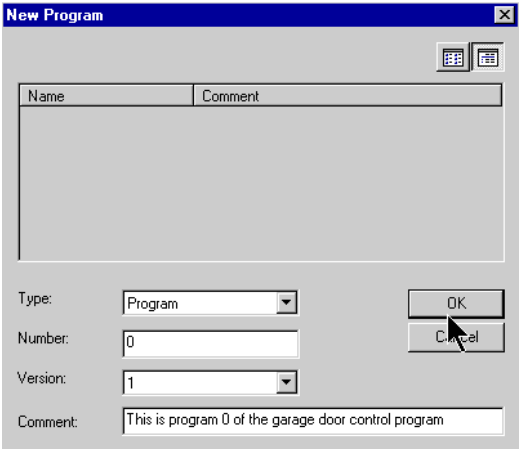



Fig. 3/16

Type: Program oder CMP or CFM – here it remains at the program
Number: 0 ... 63 – here it remains 0
Version: 1 ... 9 – here it remains 1
Comment: Garage door controller

- Clicking OK will open the program window which is still empty. Should the 'shortcut' not be visible to you, then you can open it by clicking Shortcuts in the View menu.

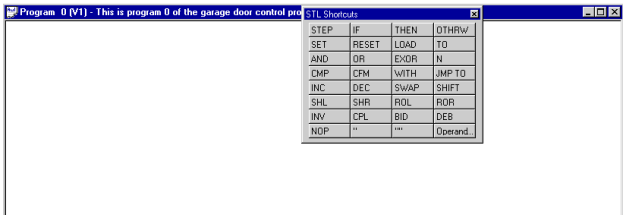



Fig. 3/17

Now the garage door controller should be described.
When should the door open?
The door should open if either

– the button is pressed inside **OR**

– the button is pressed outside **AND** the key switch is actuated at the same time.

You can use nearly the same words in the FST statement list:

```
Program 0 (V1) - This is program 0 of the garage door control program*
IF                  Open_in         'I0.2: Push button inside Open
      OR        (   Open_out        'I0.4: Push button outside Open
      AND           Key         )   'I0.6: Key switch outside
THEN  SET           Relay_op        'O0.0: K1: Relay to open the door
```

Fig. 3/18

In doing so you can make almost all the entries using shortcuts and the mouse or typing on the keyboard (or a mix of any of these).

Even the use of the symbols is released in comparison with the absolute operand address. The following program is therefore equivalent:

```
IF                I0.2     'Open_in: 'Push button inside Open
       OR       (  I0.4     'Open_out: 'Push button outside Open
       AND         I0.6  )  'Key: Key switch outside
THEN   SET         O0.0     'Relay_op: V1: Relay to open the door
```

Fig. 3/19

With the line THEN SET O0.0, though we switched on the motor to open, it has not been switched off so far. When should the motor be switched off?
The motor should be switched off when the door is open.

```
IF            open          'I0.0: Limit keys garage door is open
THEN   RESET  Relay_op      'O0.0: V1: Relay to open the door
```

Fig. 3/20

When closing the door, the respective push button should remain actuated continuously for security reasons.
We could therefore word this as follows:
The door closes **if** CLOSE is pressed on the inside **or** the outside **and** the door is not closed. **Otherwise** it will remain in place.

```
IF            (  Close_in     'I0.3: Push button inside Close
       OR        Close_out  ) 'I0.5: Push button outside Close
       AND    N  closed       'I0.1: Limit keys garage door is closed
THEN   SET       Relay_clo    'O0.1: V2: Relay to close the door
OTHRW  RESET     Relay_clo    'O0.1: V2: Relay to close the door
```

Fig. 3/21

Now all the automation technology experts will be telling us that we have forgotten some general conditions. First of all, we must make sure that the motor cannot operate in both directions at the same time. Consequently, for both directions the opposite direction must be queried along with it.
Second, we must ensure that the motor remains in place when CLOSE and OPEN are pressed at the same time.

The final program could look like this:

```
IF                Open_in     'I0.2: Push button inside Open
       OR      (  Open_out    'I0.4: Push button outside Open
       AND       Key       ) 'I0.6: Key switch outside
       AND    N  Relay_clo   'O0.1: V2: Relay to close the door
       AND    N  Close_in    'I0.3: Push button inside Close
       AND    N  Close_out   'I0.5: Push button outside Close
       AND    N  open        'I0.0: Limit keys garage door is open
THEN   SET       Relay_op    'I0.0: V1: Relay to open the door
IF                open        'I0.0: Limit keys garage door is open
       OR        Close_in    'I0.3: Push button inside Close
       OR        Close_out   'I0.5: Push button outside Close
THEN   RESET     Relay_op    'I0.0: V1: Relay to open the door
IF             (  Close_in    'I0.3: Push button inside Close
       OR        Close_out ) 'I0.5: Push button outside Close
       AND    N  closed      'I0.1: Limit keys garage door is closed
       AND    N  Relay_op    'O0.0: V1: Relay to open the door
       AND    N  Open_in     'I0.2: Push button inside Open
       AND    N  Open_out    'I0.4: Push button outside Open
THEN   SET       Relay_clo   'O0.1: V2: Relay to close the door
OTHRW  RESET     Relay_clo   'I0.1: V2: Relay to close the door
```

Fig. 3/22

Even when it looks so nice on paper, one very important component is still missing: The commentary. A program is only really 'good' when colleagues will still understand in 2 years what is intended now. That is why a program should have sound comments.
Perhaps the following would be better:

```
""Project:     Garage
""Author:      Bernhard Plagemann
""Date:        1 October 2015"

""Open garage door
IF                Open_in       'I0.2: Push button inside Open
       OR      (  Open_out      'I0.4: Push button outside Open
       AND       Key       )   'I0.6: Key switch outside
       AND    N  Relay_clo     'O0.1: V2: Relay to close the door
       AND    N  Close_in      'I0.3: Push button inside Close
       AND    N  Close_out     'I0.5: Push button outside Close
       AND    N  open          'I0.0: Limit keys garage door is open
THEN   SET       Relay_op      'O0.0: V1: Relay to open the door

""Stop garage door
IF                open          'I0.0: Limit keys garage door is open
       OR         Close_in      'I0.3: Push button inside Close
       OR         Close_out     'I0.5: Push button outside Close
THEN   RESET      Relay_op      'O0.0: V1: Relay to open the door

""Closing the garage door
IF             (  Close_in      'I0.3: Push button inside Close
       OR         Close_out )   'I0.5: Push button outside Close
       AND    N  closed        'I0.1: Limit keys garage is closed
       AND    N  Relay_op      'O0.0: V1: Relay to open the door
       AND    N  Open_in       'I0.2: Push button inside Open
       AND    N  Open_out      'I0.4: Push button outside Open
THEN   SET       Relay_clo     'O0.1: V2: Relay to close the door
OTHRW  RESET     Relay_clo     'O0.1: V2: Relay to close the door
```

Fig. 3/23

Here are some more instructions for making entries in the statement list:

- You can type the statement list like you would a letter to the controller. As soon as you press ‹Enter›, your program is formatted and the comments will be added to the allocation list.
- There must be at least one blank space between each operation and each operand.
- No blank spaces are allowed in names, operands or operations.

– When you type something that FST does not recognise, it is then assumed that you would like to define a new operand. You will then receive the following query:

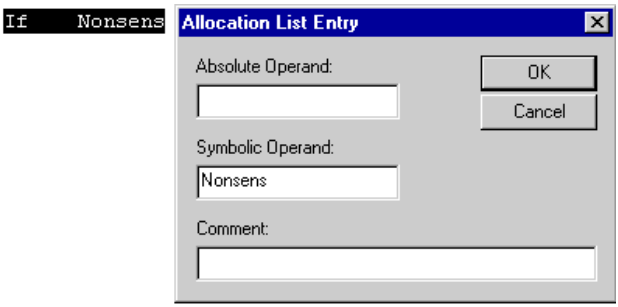



Fig. 3/24

FST presumes that 'Nonsens' is the symbol for a new operand. Simply click Cancel (or press ‹Esc›), make the correction and continue.

– FST requires Boolean sets as a matter of principle. A Boolean set consists of IF and THEN, and can possibly consist of OTHRW in addition. IF; THEN and OTHRW may each only occur exactly one time in a set.

## Program as a ladder diagram [8)]

Once the allocation list has been created, it is finally time to write the program. Every FST project requires a program with the No. 0.
Right-clicking on the Programs directory in the project window opens the context menu with the possibility of inserting a new program. Or use the icon for 'New Program' or the Insert menu.


Fig. 3/25

Now choose the programming language you want. Please note that you cannot automatically translate between the statement list and the ladder diagram. Specifying one of the two languages thus applies to the entire program. On the other hand, programs with various programming languages can be mixed within a project – Program 0 could therefore be written in statement list (STL), while Program 12 is in ladder diagram (LDR).

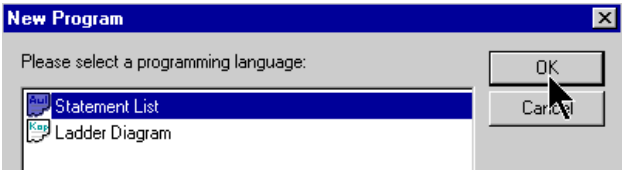



Fig. 3/26

8) You will find this project as Gar_en_l.ZIP on your CD.

The following entries are now possible:

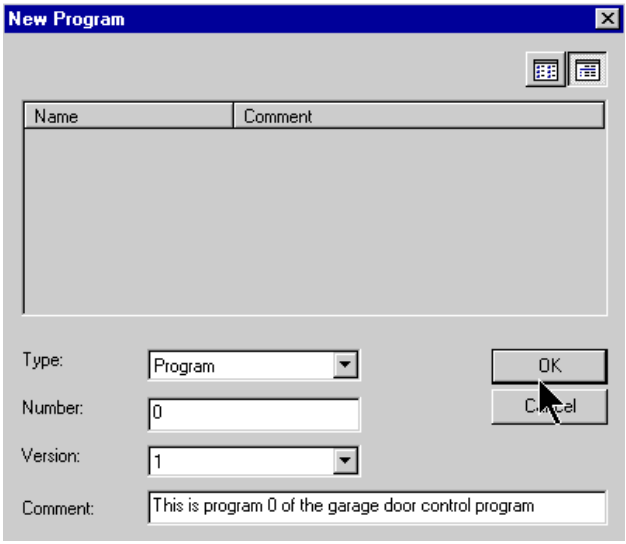



Fig. 3/27

Type: Program or CMP or CFM – here it remains Program
Number: 0 ... 63 – here it remains 0
Version: 1 ... 9 – here it remains 1
Comment: Garage door controller

Clicking OK will open the program window which is still empty. Should the 'shortcut' not be visible to you, then you can open it by clicking Shortcuts in the View menu.

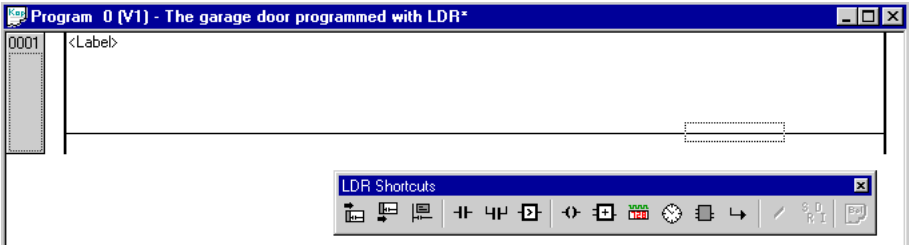



Fig. 3/28

Now the garage door controller should be described.
When should the door open? The door should open **if** either

- the button is pressed inside **OR**
- the button is pressed outside **AND** the key switch is actuated at the same time.

In the ladder diagram, an OR becomes a parallel connection, an AND a series connection and a NOT a breaker.


Fig. 3/29

Even the use of the symbols is released in comparison with the absolute operand address. The following program is therefore equivalent:

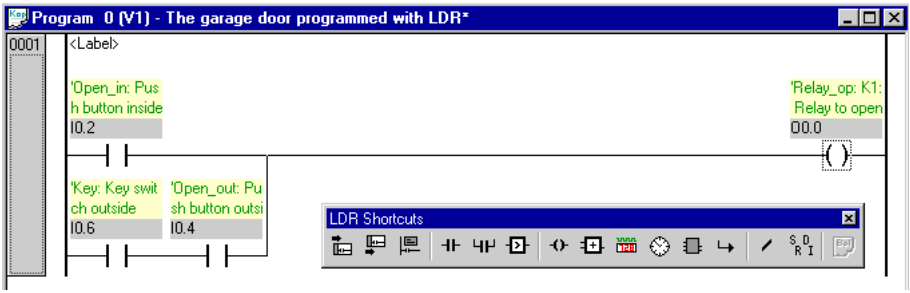



Fig. 3/30

However, our program still has a major error. Though the door does begin to open, it stops right away if one of the buttons involved is no longer actuated – a truly inconvenient way of opening the garage.

What is missing is 'locking', the memory of the ladder diagram. In addition, you must define how the locking will be set and deleted. It is set with the condition described above and deleted as soon as the garage door is open. The program will then look like this.

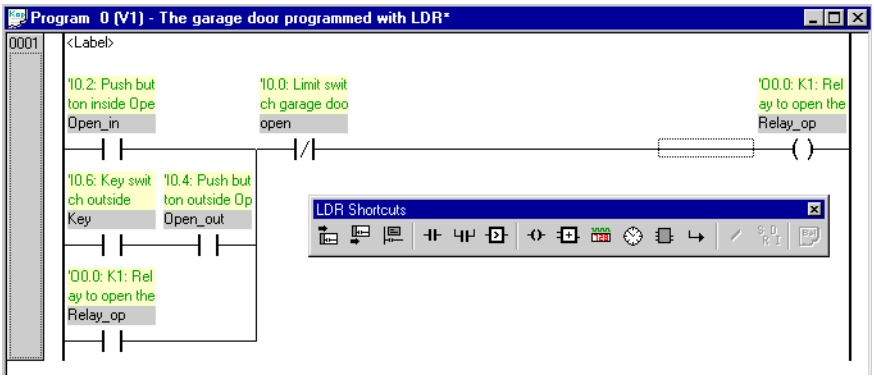



Fig. 3/31

**A few more instructions to operate the ladder diagram editor**

A ladder diagram is subdivided into networks. An individual network consists of 'contacts' und 'coils'.
Contacts and coils can be negated.
Contacts can be switched in series and in parallel.

– New networks are either inserted before or after the current network.

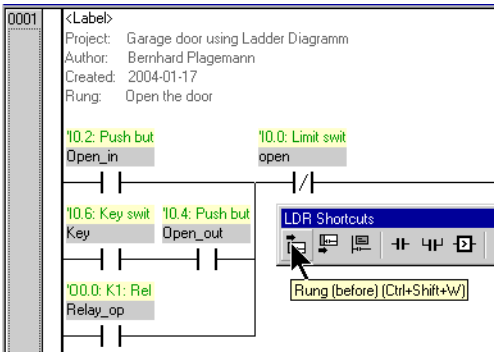



Fig. 3/32

– An additional contact in series to an existing contact is inserted by marking the current contact and inserting another one before it using the Contact icon of the short-cuts.

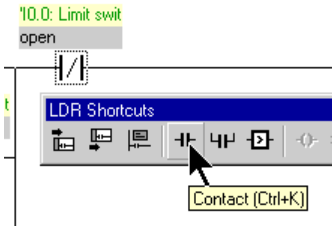



Fig. 3/33

– A parallel contact is added by marking the contact(s) to which the new contact should be parallel. The shortcuts are subsequently used.

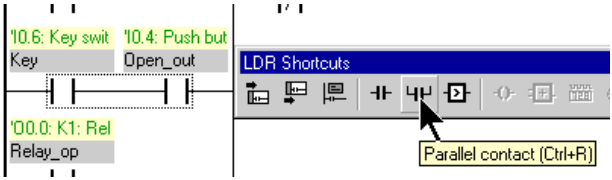



Fig. 3/34

– Another parallel coil is added if the existing coil is marked and another is added using the shortcuts.

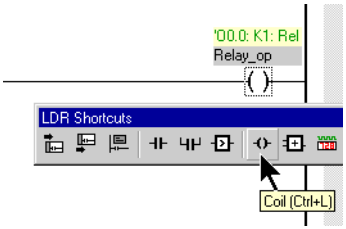



Fig. 3/35

– A contact or a coil is negated (or the negation is undone) by marking it and negating it using the shortcuts.

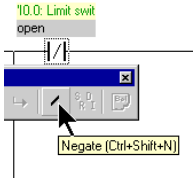



Fig. 3/36

– A coil is provided with other controller attributes (S – set, R – restore, I – increment, count forwards, D – decrement, count backwards) by marking it and then rotating through the possibilities by means of the shortcuts and clicking the one you want until it has been entered.

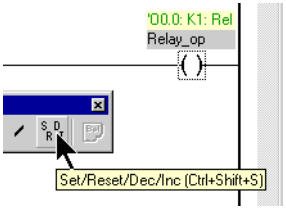



Fig. 3/37

– The comments on the operands are displayed in two lines in these examples. They can be set in the menu Extras → Preferences... → LDR → Lines for Operand Comments.

– When you type something as an operand that FST does not recognise, it is then assumed that you would like to define a new operand. You will then receive the following query:

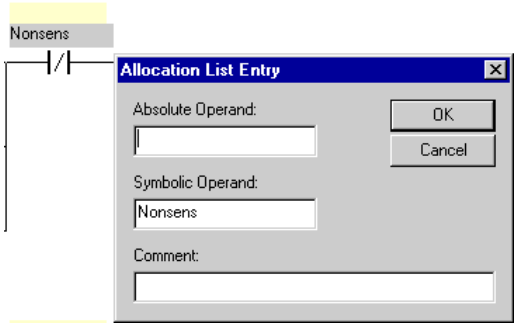



Fig. 3/38

– FST presumes that 'Nonsens' is the symbol for a new operand. Simply click Cancel (or press ‹Esc›), make the correction and continue.

Back to the garage door:
When closing the door, the respective push button should remain actuated continuously for security reasons.
We could therefore word this as follows:
The door closes **if** CLOSE is pressed on the inside **or** the outside **and** the door is not closed. **Otherwise** it will remain in place.


Fig. 3/39

Now all the automation technology experts will be telling us that we have forgotten some general conditions. First of all, we must make sure that the motor cannot operate in both directions at the same time. Consequently, for both directions the opposite direction must be queried along with it. Second, we must ensure that the motor remains in place when Close and Open are pressed at the same time.

The final program could look like this:

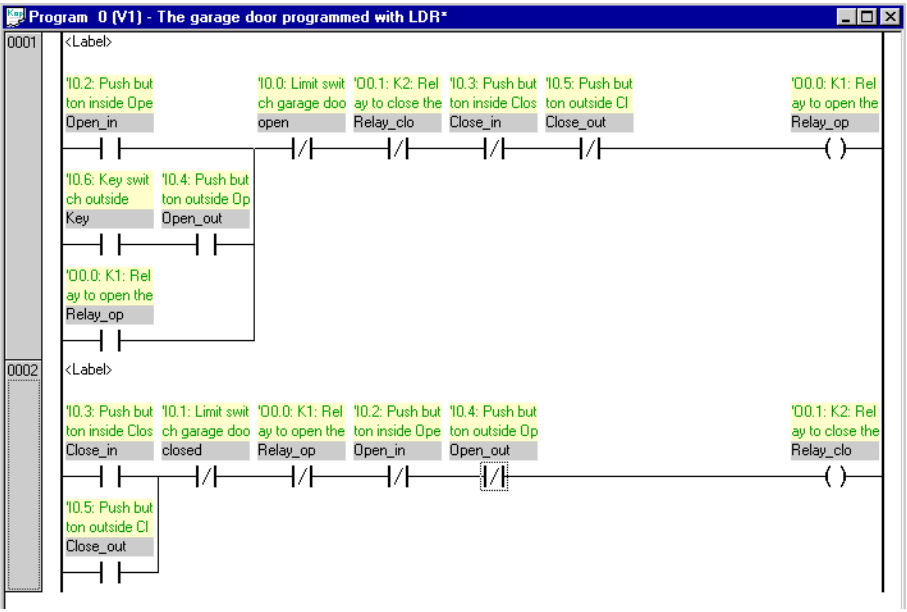



Fig. 3/40

Even when it looks so nice on paper, one very important component is still missing: The commentary. A program is only really 'good' when colleagues will still understand in 2 years what is intended now. That is why a program should have sound comments.
Perhaps the following would be better:

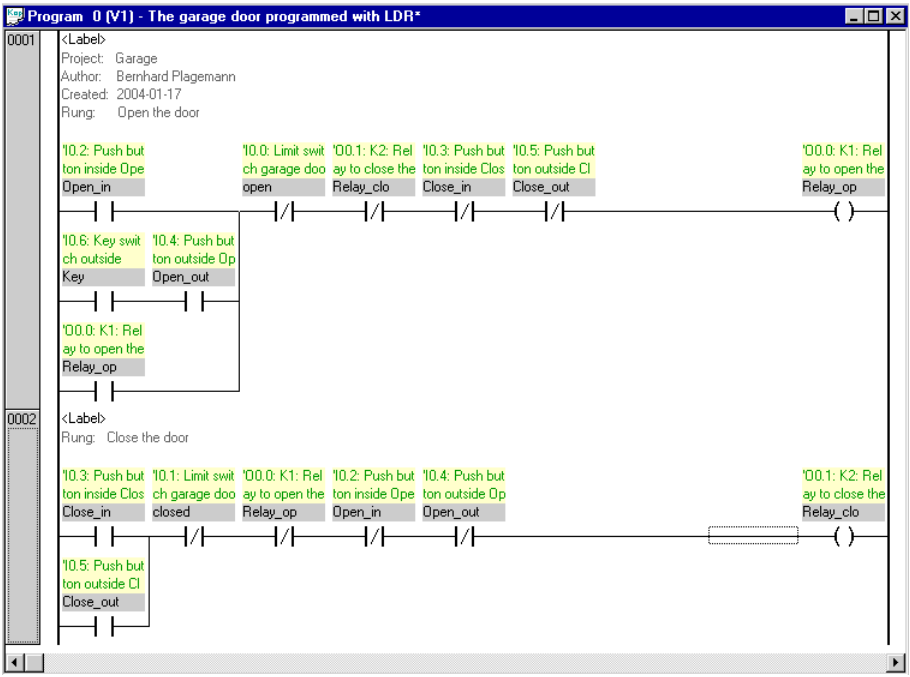



Fig. 3/41

## 3.1.5 Compiling

The FST software does not always agree with everything we write. A syntax check, which searches the program for formal errors, is performed during compilation. Click the icon for 'Build Project'.

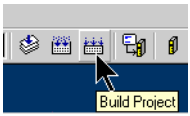
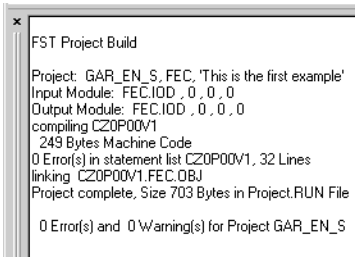



Fig. 3/42

You will receive a message about the check and the compilation of the statement list or the ladder diagram.



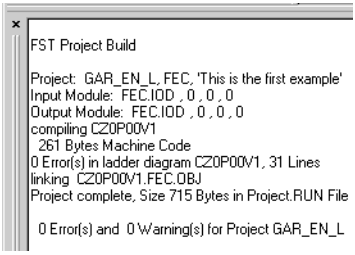




Fig. 3/43

## Errors in the statement list [9)]

This won't always work out so nicely on the first try.
The following example shows an error:

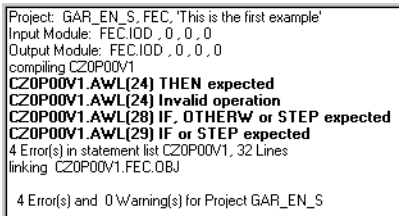



Fig. 3/44

FST reports 4 errors in this example. The 4 errors should be found in:

CZ0P00V1.AWL[24] THEN expected
and
CZ0P00V1.AWL[24] THEN Invalid operation
and
CZ0P00V1.AWL[28] IF, OTHERW or STEP expected
and
CZ0P00V1.AWL[29] IF or STEP expected

The meaning is:
CZ0P00 – Program 00 (P00)
V1 – Version 1
[24] – in line 24

9) If you have programmed in ladder diagram, please skip to the chapter 'Errors in the ladder diagram' on the next page.

- By double-clicking the line number, e.g. [24], FST automatically skips to the specified spot.

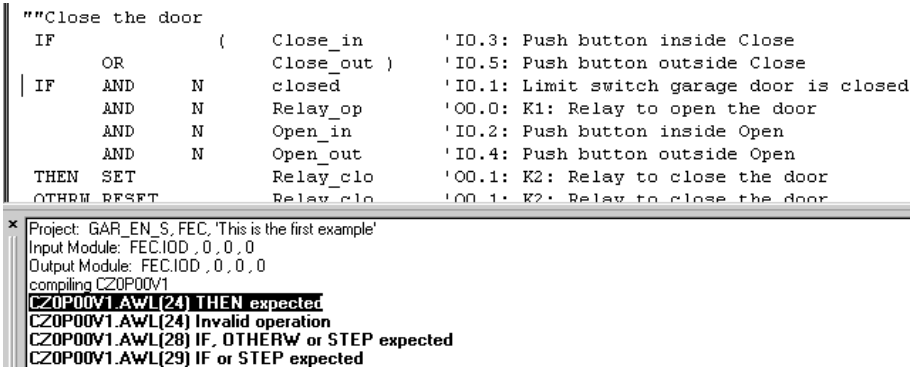



Fig. 3/45

The current line number can also be seen in the lower right of the status bar if the cursor is in the programming window.

In this example, the set beginning IF was entered twice in a Boolean set.

## Errors in the ladder diagram

This won't always work out so nicely on the first try.
The following example shows an error:

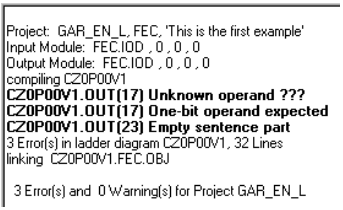



Fig. 3/46

The three errors should be found in:
`CZ0P00V1.OUT(17) Unknown operand ???`
and
`CZ0P00V1.OUT(17) One–bit operand expected`
and
`CZ0P00V1.OUT(23) Empty sentence part`

The meaning is:
CZ0P00 – Program 00 (P00)
V1 – Version 1
[17] – in line 17 of the compiled code. Double-clicking the first error opens Program 0 and positions the cursor in the network in which the error is suspected.

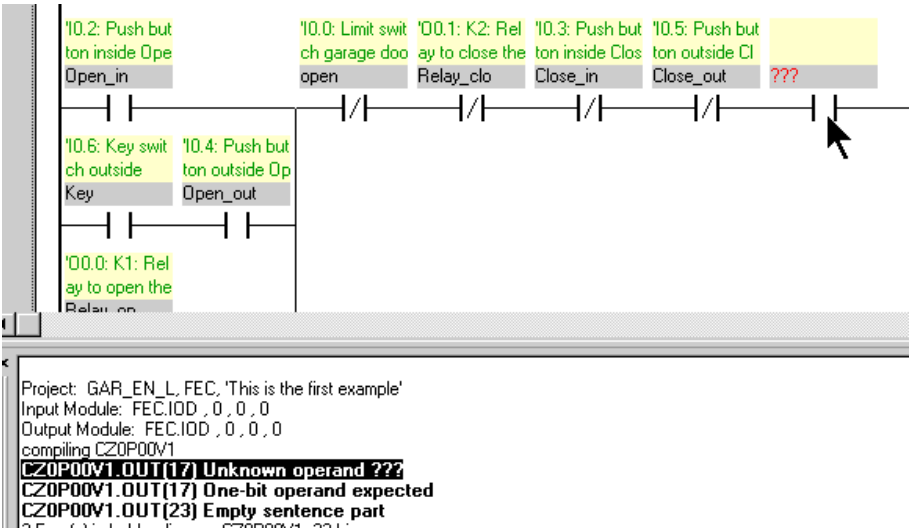



Fig. 3/47

In this case, a contact has been entered without stating the operand (input, output, flag ...).

### 3.1.6 Downloading the project

- Now please connect your controller to the power and use the programming cable to connect the controller and your programming PC.

  In our example, the FEC Compact FC20, the programming cable for the FEC is plugged into the port with the 'COM' designation; for the programming PC, the programming cable is usually plugged into COM1 or COM2.

- As soon as the controller has been connected and the programming cable is plugged in, you can test whether or not you are able to establish a connection by using the 'Control Panel' icon.

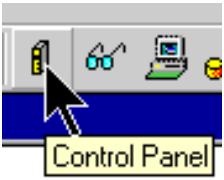



Fig. 3/48

Should a connection result, you will receive a message which could look like this:

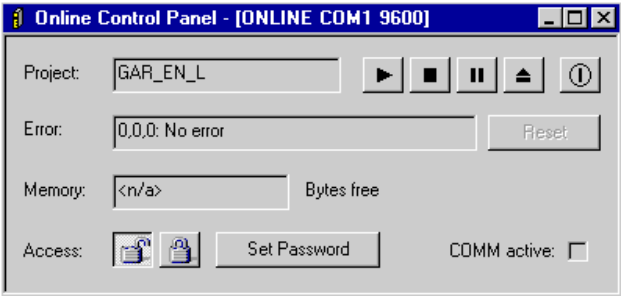



Fig. 3/49

If a connection does not result, please check

- the settings on your PC. Open the menu Extras, entry 'Preferences...', card 'Communication'. Here you can check whether the right serial port and the baudrate 9600 have been entered.

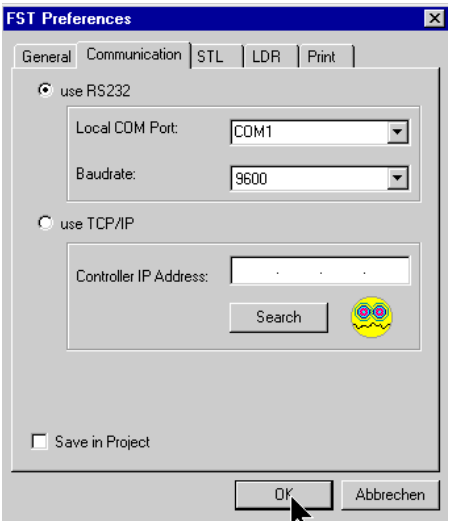



Fig. 3/50

- The Power LED must light up green on the controller and the programming cable must be plugged into the COM port. The RUN LED can light up green, orange or red, depending on how it was previously used.

If the connection was successful, then you can download the project.


Fig. 3/51

The message 'Download complete' must be given in the message window.

Depending on how your controller was previously used, you must now set the controller to RUN.

**Please note**

- Please check the RUN LED:
  If it is green, then your controller is already operating in RUN mode.
  If the LED is orange, then you must set the controller to RUN using the RUN/STOP slide switch [9].

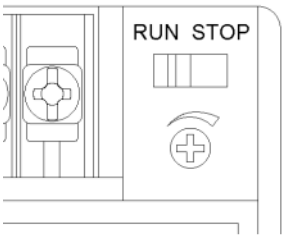



Fig. 3/52

[9] If you are using a FEC Standard, a rotary switch is used for RUN/STOP. The controller is in STOP in position 0, in RUN in all other positions.

## 3.1.7 Checking

Of course, it is of great importance that we check whether the automation project is really working the way we want it to. To do so, actuate the respective push buttons and limit keys and observe how the garage door reacts.

However, it is also important to observe what the controller is doing and how the automation projects reacts from the controller's perspective.

You have two essential possibilites in particular for this. First, you can actively observe the program, and second, the inputs and outputs.

- To observe the program, please open the window of your Program 0 and right-click in this window.

  You will receive a context menu, irrespectiv of whether you have programmed in statement list or ladder diagram, and contained in the menu in the entry 'Online'.

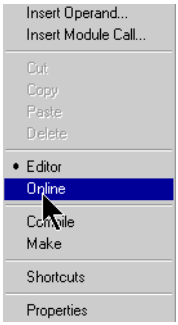



Fig. 3/53

- Activate the online mode, then the status of the individual variables will be displayed in the program window (STL) or whether the condition has been met within a rung (LDR).

```
""Open the door
 IF                    Open_in          OFF  'I0.2: Push button inside Open
       OR         (    Open_out         OFF  'I0.4: Push button outside Open
       AND           Key       )      OFF  'I0.6: Key switch outside
       AND      N    Relay_clo        OFF  'O0.1: K2: Relay to close the door
       AND      N    Close_in         OFF  'I0.3: Push button inside Close
       AND      N    Close_out        OFF  'I0.5: Push button outside Close
       AND      N    open             OFF  'I0.0: Limit switch garage door is open
 THEN  SET          Relay_op         OFF  'O0.0: K1: Relay to open the door

""Stop the door
 IF                    open             OFF  'I0.0: Limit switch garage door is open
       OR             Close_in         OFF  'I0.3: Push button inside Close
       OR             Close_out        OFF  'I0.5: Push button outside Close
 THEN  RESET          Relay_op         OFF  'O0.0: K1: Relay to open the door

""Close the door
 IF               (    Close_in         OFF  'I0.3: Push button inside Close
       OR             Close_out )      OFF  'I0.5: Push button outside Close
       AND      N    closed           OFF  'I0.1: Limit switch garage door is closed
```

Fig. 3/54

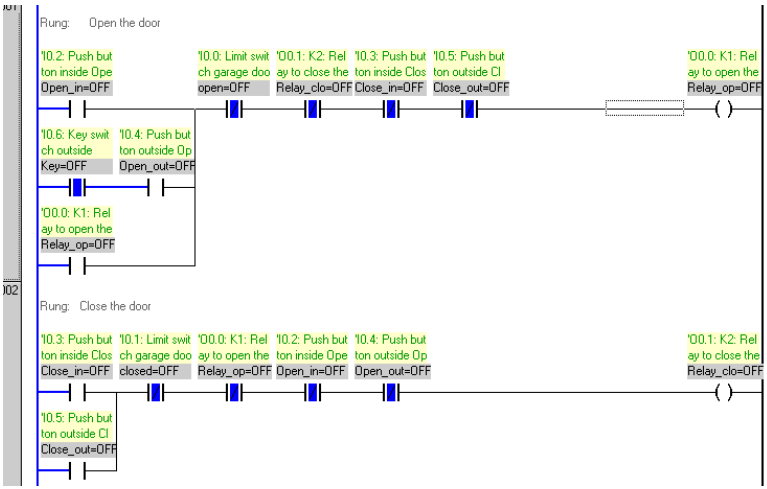



Fig. 3/55

In the program window, you will recognise the contexts in the online mode within the programm excerpt that is visible. However, it is often important to observe variables independent of the program window that is visible. Therefore, you can call up the 'Online Display'.


Fig. 3/56

Here you can observe the inputs, for example.

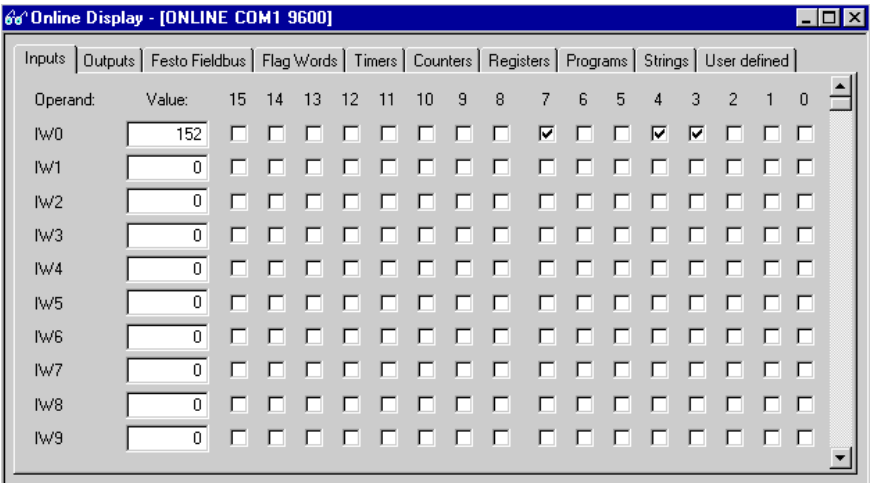



Fig. 3/57

You see in the display that the inputs I0.3, I0.4 and I0.7 are activated.

These opportunities for observation make it very easy to put a project into operation.

### 3.1.8 Documentation

Of course, nobody can take over for you the task of writing a manual for your process or machine. However, FST does allow you to handle important documentation tasks within the project. This includes:

– the comments in the program and

– the project documentation.

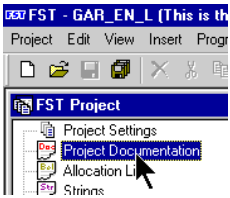



Fig. 3/58

The project documentation is normally a pure text file. If you double-click on 'Project Documentation', a file project.txt will open with the text editor activated on your Windows system. The file is stored in the project directory of your current project.

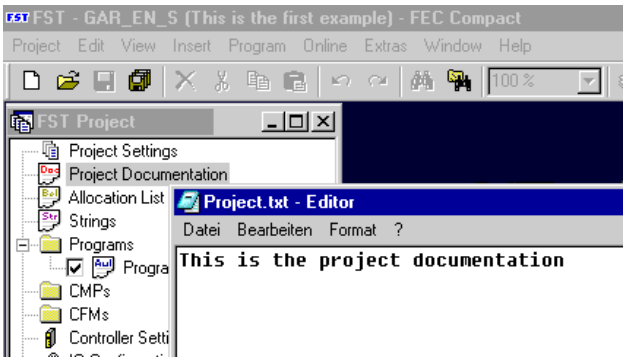



Fig. 3/59

The entire project can ultimately be printed.

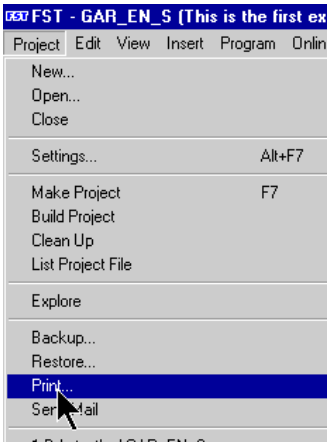



Fig. 3/60

Choose this entry; you can select the components of the project documentation that you want.

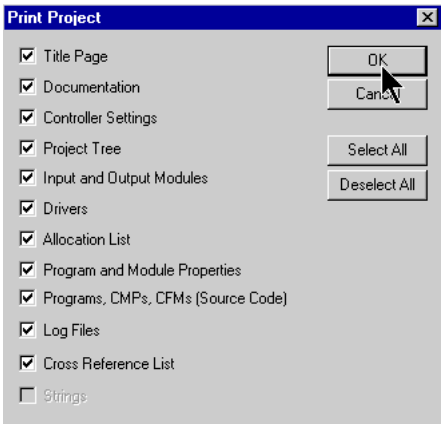



Fig. 3/61

### 3.1.9 The Garage project

If you also installed sample files when you installed FST, the sample projects are already located the FST project directory. For the Garage project, they are:

– Gar_en_s (programmed in statement list)

– Gar_en_l (programmed in ladder diagram).

The sample projects from this book are archived on the FST CD in the subdirectory \Samples\Deutsch or \Samples\English. If necessary, you can read them using FST (restore). Select the command Restore in the Project menu.

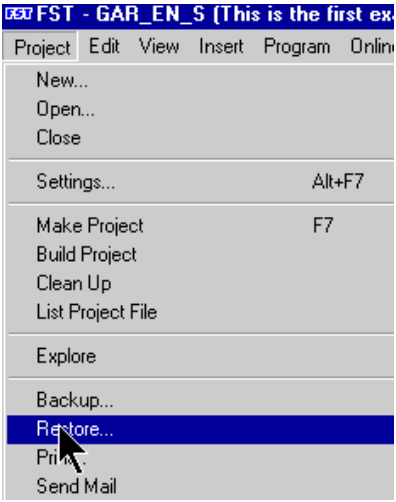



Fig. 3/62

In the following dialog, look for the file you want (e.g. GAR_EN_S.ZIP or GAR_EN_L.ZIP) on the CD in the subdirectory \Samples\Deutsch or \Samples\English.

## 3.2 Summary

Here are most important points of this chapter in a nutshell.

**Please note**
If you are programming in FST, then create a project first. A project is necessary for each individual controller (each CPU). The method of proceeding corresponds to the following table.

| No. | What to do | Comment |
|---|---|---|
| 1 | Call up a new project | In the Project menu |
| 2 | Name the project | In the dialog box |
| 3 | Select the proper controller, enter project comment | You can use FST software to program completely different CPUs of an entire controller family. |
| 4 | I/O configuration | There will hardly ever be an automation project without any inputs and outputs. |
| 5 | Programming | At least a Program 0 is mandatory |
| 6 | Make project | You will be automatically asked to do this |
| 7 | Download the project | A connection to the controller must exist to do this |
| 8 | Check | And possibly correct, improve, optimise |
| 9 | Document | Describe, make comments, print |

Tab. 3/1

- For programming, the allocation list for the inputs and outputs is usually created first.
- FST projects always require a Program 0.
- Statement list
  - Programming is done with IF ... THEN ... OTHRW within a program.
  - IF and THEN **must** be used exactly one time in every Boolean set.
  - OTHRW **may** be used but is not necessary.
- Ladder diagram
  - Networks are programmed within a program.
  - A network consists of contacts and coils.

# A slightly larger project with FST: Controlling a drilling machine

## Chapter 4

# Contents

The example of the garage door was quite nice at the beginning, but a great deal of attributes were indeed missing which begin to make automation technology interesting. Operating modes such as automatic, manual or inching, for example, or step-by-step processing of a program and much more.

That is why this chapter should deal with a slightly more challenging project. The drilling machine

1 Clamp cylinder

2 Drill cylinder

3 Ejector

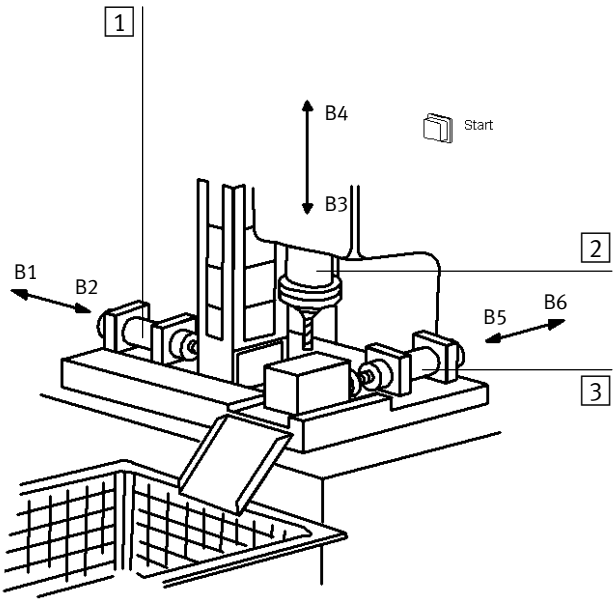



Fig. 4/1

Let's define some general conditions, as is always the case in an automation project.

## 4.1 The drilling machine

### 4.1.1 Operation

The drilling machine is operated via a control panel. Here you will find the following buttons:

| Button | Function |
|---|---|
| Start | Starts the automatic sequence |
| Halt | Stops the sequence at the end of a cycle (breaker) |
| Continuous/ inching mode | Selector switch that switches between automatic and inching mode. The Start button is used in inching mode to continue switching step-by-step |
| Stop | Stops the sequence immediately (breaker) |
| Automatic/ manual | Selector switch that switches between automatic and manual mode. Manual mode can only be switched on when automatic mode is not active. |
| Back to home position | Returns the machine to home position Back to home position can only be started if the manual mode is active. |
| Clamp | Button for the clamp cylinder in manual mode |
| Drill | Button for the drill cylinder in manual mode |
| Eject | Button for the ejector cylinder in manual mode |
| Emergency stop | Emergency stop switches the controller's outputs without stress. In addition, emergency stop sends a signal to an input so the controller knows about the emergency stop. After releasing the emergency stop, the system must be returned to home position using Back to home position before it can be restarted. |

Tab. 4/1

The indicator lights can be used:

| Button | Function |
|---|---|
| Start light | Lights up when the system is in home position and automatic mode can be started using Start. |
| Back to home position light | Lights up as long as the home position program is active |
| Emergency light | Shows if Emergency stop is or was actuated until the machine is again in home position after the emergency stop |
| Automatic light | Shows if the machine is operating in continuous mode |

Tab. 4/2

### 4.1.2 Actuators and sensors

The drilling machine has three actuators: Clamp cylinder, drill cylinder and ejector. If the cylinders are allocated letters in the traditional way, they are then cylinders A, B and C. Each cylinder is equipped with two limit keys, one to clamp and the other to release.

### 4.1.3 The program

The three cylinders must operate in the following movement sequence in automatic mode:
A+ B+ B- A- C+ C-

The movement sequence is described as follows in the function plan (is not generated in FST, this chart is merely an example):

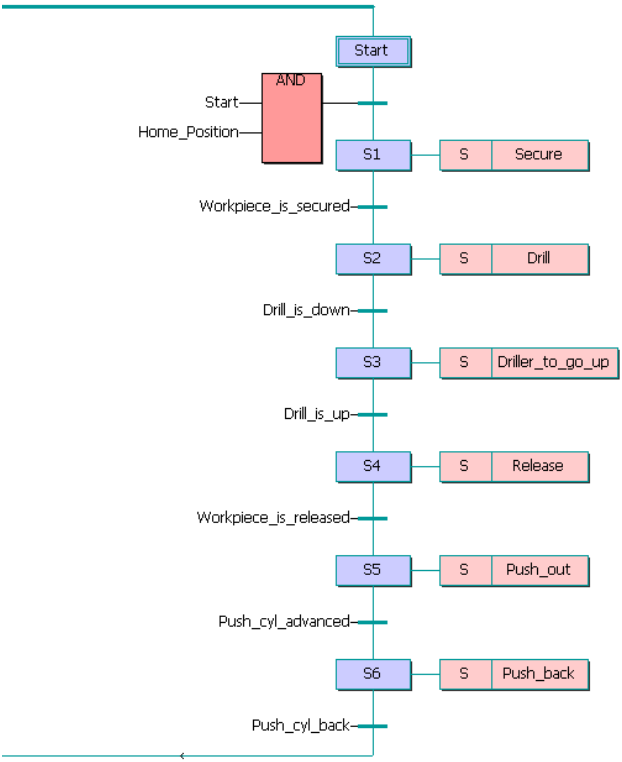



Fig. 4/2

### 4.1.4 Allocation

The allocation list requires 16 inputs and 8 outputs. A FEC Standard FC440 is used as a example controllller. The entire allocation list could look like this. [10)]

```
Operand Symbol          Comment

I0.0    Released        B1: Clamp cylinder is released
I0.1    Clamped         B2: Clamp cylinder is clamped
I0.2    Drill_dow       B3: Drill cylinder is down
I0.3    Drill_up        B4: Drill cyllinder is up
I0.2    Ejected         B5: Ejector is forward
I0.5    Ej_back         B6: Ejector cylinder is back
I0.6    Start           S1: Start button
I0.7    Halt            S2: Halt button (breaker)
I0.8    Continous       S3: Continuous mode switch (0=inching mode)
I0.9    Stop            S4: Stop button (breaker)
I0.10   Automatic       S5: Automatic switch (0=manual)
I0.11   Home_pos        S6: Back to home position button
I0.12   Man_CylA        S7: Clamp cylinder manual button
I0.13   Man_CylB        S8: Drill cylinder manual button
I0.14   Man_CylC        S9: Ejector manual button
I0.15   Em_Stop         S10: Emergency stop switch (breaker)
O0.0    Aplus           Y1: Clamp the workpiece
O0.1    Aminus          Y2: Release the workpiece
O0.2    Driller         Y3: Drill cylinder
O0.3    Ejector         Y3: Ejector
O0.4    L_Start         H1: Start light
O0.5    L_Home          H2: Back to home position light
O0.6    L_Em            H3: Emergency stop light
O0.7    L_Auto          H4: Automatic continuous mode light
```

Fig. 4/3

[10)] A note on proceeding: I created the allocation list in an Excel table. This allows for simple copying and changing. Then I completely marked, copied and pasted the table in FST. In so doing, FST checks for correct syntax, in this example invalid lengths, for example. The symbol may have no more than 9 characters, the comment no more than 36.

## 4.2 Program organisation

To ease programming, it is commonly accepted these days that you divide a project into 'sections', 'blocks', 'modules' or – as in FST – into **programs**.

Please observe the following terms:
Everything having to do with an individual central unit (or an individual central controller) is named 'Project'.
There are 'programs' with the project, exactly up a maximum of 64 programs numbered from 0 to 63, therefore P0 ... P63.
The program PO plays a special role. It is the program that is automatically started when the controller is switched on. If it is not available, the controller sends an error message and switches to Stop.

All the programs can work simultaneously; the term here is 'multitasking'.

To organise the 'Drilling machine' project, you could organise the programs as follows:

P0 Organisation program
P1 Automatic program and inching
P2 Back to home position
P3 Recognising and treating errors in the machine
P4 Manual program

This results in a project window as follows:

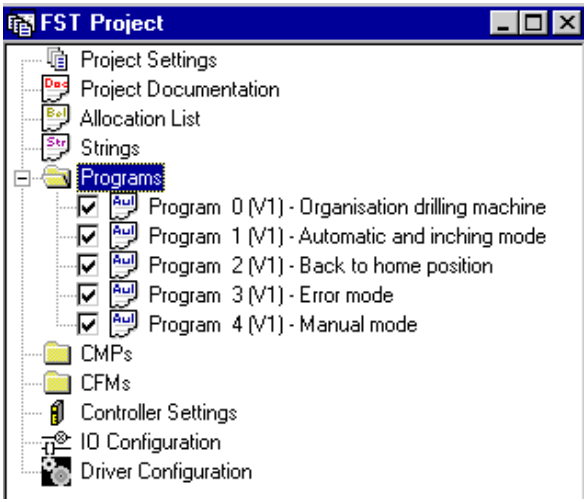



Fig. 4/4

The allocation list is also expanded:

```
Operand Symbol  Comment
P0              Organisation program
P1              Automatic program
P2              Back to home position
P3              Error mode
P4              Manual mode
```

Fig. 4/5

## 4.3 The first steps in the statement list [11/12)]

So that the 'Drilling machine' project can gradually come about, individual tasks should be fulfilled consecutively. To be able to 'see' anything at all, a simple automatic sequence should be programmed first. We need programs P0 and P1 to do this.

### 4.3.1 Starting the program

The automatic program should be temporarily started in program P0 without any conditions. It only needs two lines to do this [13)]:

```
IF                    NOP
THEN    SET           P1
```

Fig. 4/6

It is also possible to switch on (SET) [14)] and off (RESET) a program just as you would an output.

11) You will find this first drilling machine project as Drill_01.ZIP on your CD.

12) If you would like to program in ladder diagram, skip to section 4.4.

13) Note for those adept in FST software for FPC controllers: There is no longer an IF NOP THEN PW. The PW operation has been omitted.

14) Note for those adept in FST software for FPC controllers: The query 'IF N PX THEN SET PX' is no longer necessary. The command SET PX does not cause a return jump in active PX.

## 4.3.2 Sequential program

The automatic program is a program in which one step is processed after the other. The FST software provides the STEP operation for this type of program, the sequence program. We can therefore program the automatic sequence quite simply:

```
"" Automatic mode of the drilling machine

STEP Aplus
 IF              Released     'B1 to I0.0 Clamp cylinder is released
       AND       Bohr_up      'B4 to I0.3 Drill clinder is up
       AND       Ej_back      'B6 to I0.5 Ejector cylinder is back
       AND       Em_Stop      'S10 to I0.15 Emergency stop (breaker)
       AND       Stop         'S4 to I0.9 TStop switch (breaker)
       AND       Start        'S1 to I0.6 Start button
 THEN  RESET     Aminus       'Y2 to O0.1 Clamp cylinder back
       SET       Aplus        'Y1 to O0.0 Clamp the workpiece
STEP Bplus
 IF              Released     'B2 to I0.1 Clamp cylinder is forward
 THEN  SET       Driller      'Y3 to O0.2 Drill cylinder
STEP Bminus
 IF              Drill_dow    'B3 to I0.2 Drill clinder is down
 THEN  RESET     Driller      'Y3 to O0.2 Drill cylinder
STEP Aminus
 IF              Bohr_up      'B4 to I0.3 Drill clinder is up
 THEN  RESET     Aplus        'Y1 to O0.0 Clamp the workpiece
       SET       Aminus       'Y2 to O0.1 Clamp cylinder is back
STEP Cplus
 IF              Released     'B1 to I0.0 Clamp cylinder is released
 THEN  SET       Ejector      'Y4 to O0.3 Ejector
STEP Cminus
 IF              Ejected      'B5 to I0.4 Ejector cylinder is forward
 THEN  RESET     Ejector      'Y4 to O0.3 Ejector
       JMP TO Aplus
```

Fig. 4/7

### 4.3.3 Downloading the project

If you would like to test the first simple version, the programs P2, P3 and P4 must be deactivated before you can compile. Although they are created, they have not yet been used and are still empty.
Simply use the mouse to click in the appropriate box.

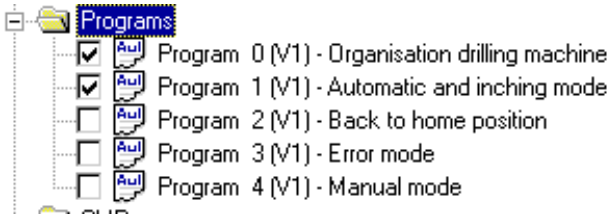



Fig. 4/8

## 4.4 The first steps in ladder diagram [15)]

So that the 'Drilling machine' project can gradually come about, individual tasks should be fulfilled consecutively. To be able to 'see' anything at all, a simple automatic sequence should be programmed first. We need programs P0 and P1 to do this.

### 4.4.1 Starting the program

The automatic program should be temporarily started in program P0 without any conditions. You only need a network to do this:

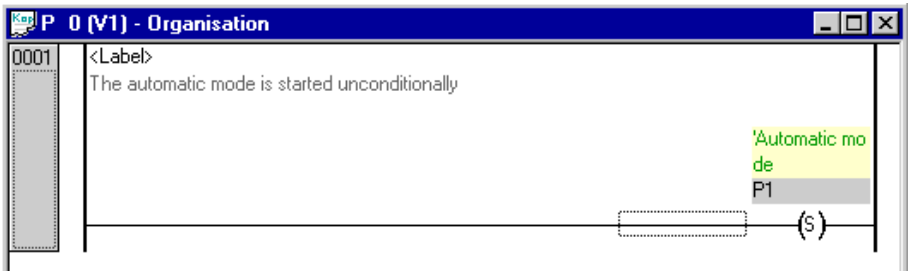



Fig. 4/9

It is also possible to switch on (S) [16)] and off (R) a program just as you would an output.

**Please note**
Kindly observe that program P1 is set here with the attribute S. An assignment (coil without attribute) is invalid for programs (P...).

15) You will find this first drilling machine project as Drill_L1.ZIP on your CD.

16) Note for those adept in FST software for FPC controllers: The query 'IF N PX THEN SET PX' is no longer necessary. The command SET PX does not cause a return jump in active PX.

## 4.4.2 Sequential program

The automatic program is a program in which one step is processed after the other. A step structure is simulated in the ladder diagram either with a counter or with flags. Programming with flags will very likely occur more often (even if it is not more comfortable).

### Step programming with flags

Every step receives a simple self locking. The self locking is set by an AND link from the transition condition for this step and the query whether the previous step is active.

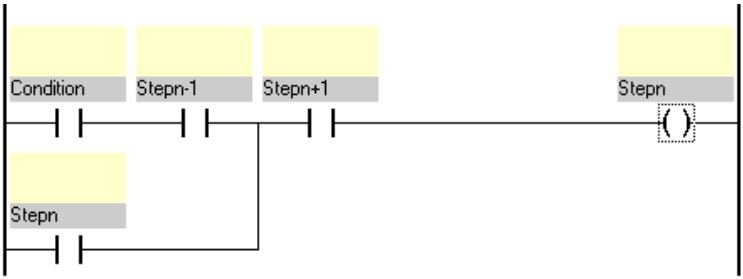



Fig. 4/10

Both the first and the last step each have a special feature.

– In the first step, there is not a transition condition. Therefore, the start condition must be worded precisely enough that the controller cannot be accidentally started a second time in the middle of the automatic sequence. In case of doubt, critical steps must be queried as to whether or not they are active, if necessary.

– In the last step, there is not a next step. Therefore, the last step is not programmed as locking. It is automatically deleted as soon as the previous one has been deleted. However, here you must take care that only a single CPU cycle is available to actions of the last step. If necessary, another step must be programmed to make sure that the last step can actively cause something.

Finally, the controller's outputs are switched at the end of the program. In the process it is investigated for each output in which steps it must be active.

The program could look like this:

Sequencer for steps 1 to 3

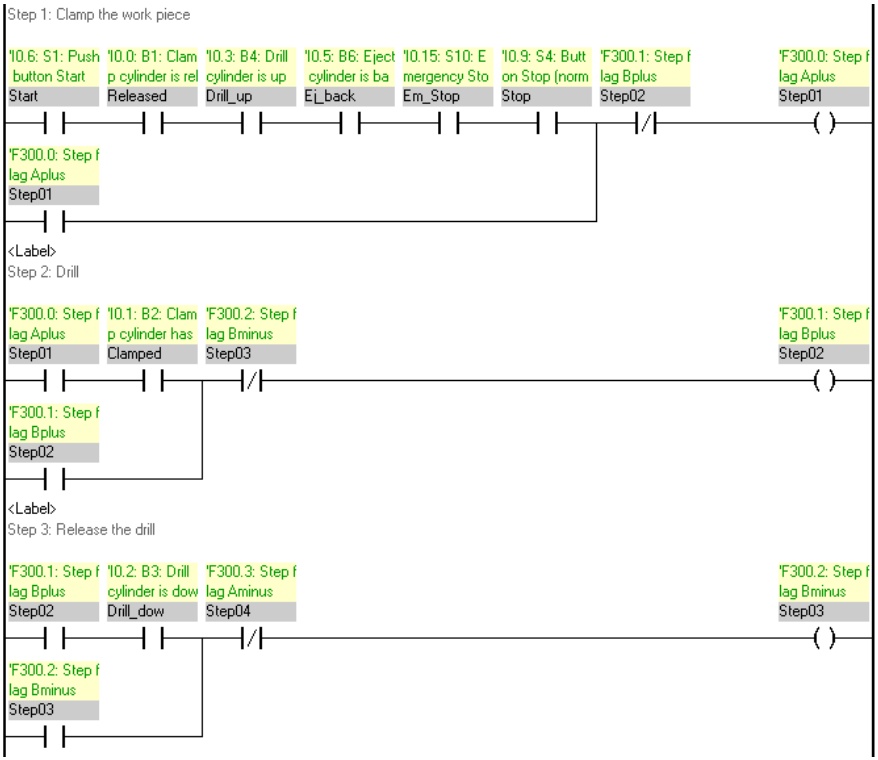



Fig. 4/11

Sequencer for steps 4 to 6

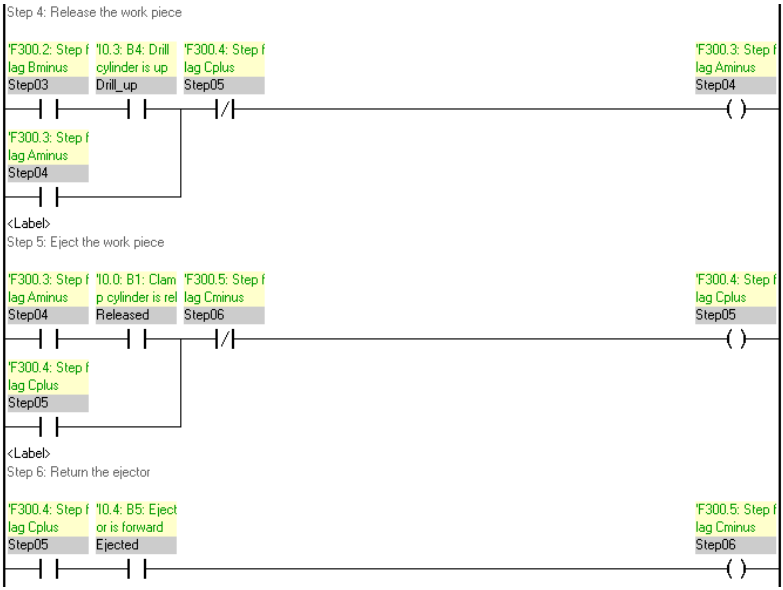



Fig. 4/12

Switching outputs (cut-out)


Fig. 4/13

**Please note**

– If the flags used in such a program are remanent [17], the last active step will remain in the event of power loss. If you wish to do so, then the warm recovery after the power supply is restored must be planned precisely.

– If the flags used are not remanent, they will lose their content when the power supply is switched off; the program will then always begin with the first step after being switched on. In the above example, the flags 300.0 and following have been chosen deliberately, as this flag range is not remanent for the FEC controllers (FEC Compact, FEC Standard).

[17] Remanence designates secure behaviour of a PLC operand in the event of power loss. In the event of loss of electric energy, the flags receive the most recent status, thus 0 or 1. Non-remanent or volatile operands become 0 when they are not supplied with energy.

### 4.4.3 Step programming with counters

If a counter [18] is used instead of flag bits, then the counter status' query is the first part of the every step. A step looks like this:


Fig. 4/14

The jump to certain step occurs by loading a number in the step counter, as with a return jump at the beginning.

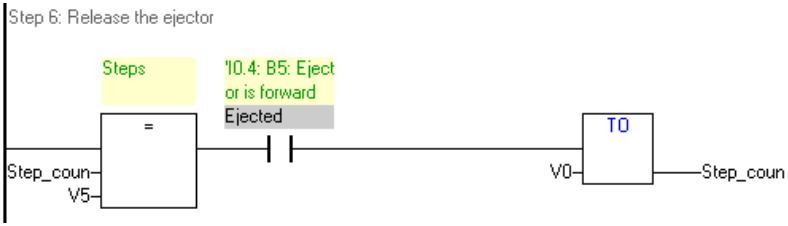



Fig. 4/15

[18] You will find this program example as Drill_L2.ZIP on your CD.

### 4.4.4 Downloading the project

- If you would like to test the first simple version, the programs P2, P3 and P4 must be deactivated before you can compile. Although they are created, they have not yet been used and are still empty.
- Simply use the mouse to click in the appropriate box.

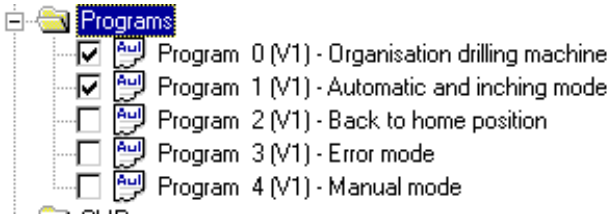



Fig. 4/16

# The STEP operation in the statement list

## Chapter 5

# Contents

In the previous chapter we already used the operation STEP, but largely intuitively, that is, without exactly knowing why which step causes something. In this chapter we want to take a closer look at the mysterious STEP [19] operation.

So here are some rules.
As soon as the STEP operation is used in a program

- exacly one single step is always active in this program
- one step is taken after the other if 'something can be done' in the last Boolean set (IF ... THEN ...OTHRW). This means that it will go to the next step if
  - in a Boolean set consisting of only IF ... THEN the IF part is fulfilled or
  - in a Boolean set consisting of IF ... THEN ... OTHRW, the THEN part (fulfilled condition) or the OTHRW part (non-fulfilled condition) was processed.
- you can jump between steps using the operation 'JMP TO'
- a jump must be defined in the last step or the program will be terminated.

[19] Ladder diagram programmers can skip this chapter without having to substitute another one for it.

## 5.1 The STEP

The basic support provided by the Step operation in programming is just this: If there are steps in a program, only a single step can be active at a time, all the other steps will not be observed. Imagine a step program with 170 steps and 1800 lines: Provided that the program itself is active (see the chapter 'Multitasking'), then only that section of the 1800 lines of program code is processed which is part of the step currently active.

Consider the step Aminus in the automatic program.
The following is asked in this step:

```
STEP Aminus
 IF           Drill_up       'B4 to I0.3 Drill cylinder is up
 THEN  RESET     Aplus       'Y1 to O0.0 Clamp the workpiece
       SET       Aminus      'Y2 to O0.1 Clamp cylinder is back
```

Fig. 5/1

The condition Drill_up, that is, that the drill is up, is also fulfilled in the step Aplus. This condition would not suffice on its own to release cylinder A again.

In addition, you must make sure that the part

– has been clamped and

– drilled.

The previous steps must therefore already be completely processed before the step Aminus can be observed in the first place.

You can imagine that the step program is no longer 1800 lines (to adhere to the example above) but only 10 or 15 lines, thus exactly as long as the current step. This applies until you go to the next step.

## 5.2 The content of a step

The usual Boolean sets can be in a step:
IF ... THEN (... OTHRW)
The operation OTHRW should be used in a step program only after consideration, as it immediately goes to the next step.

Many such Boolean sets can be in a step; the number is practically unlimited.

## 5.3 Going to the next step

Of course, it is of great importance to go from one step to the next. Going to the next step is defined as follows:

One goes from the currently active step to the next step (i.e., the currently active step becomes inactive and the next step becomes active) if 'something can be done' in the last Boolean set (IF ... THEN ... OTHRW). This means that it will go to the next step if

– in a Boolean set consisting of only IF ... THEN the IF part is met or

– in a Boolean set consisting of IF ... THEN ... OTHRW, the THEN part (fulfilled condition) or the OTHRW part (non-fulfilled condition) was processed.

Please observe: The last Boolean set of a step is always meant here!

Example:

```
STEP Example
IF                          I.0
THEN    SET                 O0.0
IF              N           I0.1
THEN    SET                 F10.0
        RESET               F100.0
IF                          F10.0
THEN    RESET               O10.5

STEP Forward
```

Fig. 5/2

In this example, only the two lines
IF F10.0
THEN RESET O10.5

are responsible for going from the Example step to the Forward step. All the other lines are executed as long as the step is active (therefore at least one time).

As long as the condition to go to the next step – this is IF F10.0 in this example – is not met, this step is processed cyclically, always beginning with the first line after the line STEP Example.

## 5.4 The name of the step

You can jump from one step to another. To do this (and only this), you need a step name. The example of the drilling machine could thus forego many step names. The example might look like this:

```
STEP Aplus
 IF            Released     'B1 to I0.0 Clamp cylinder is released
       AND     Drill_up     'B4 to I0.3 Drill clinder is up
       AND     Ej_back      'B6 to I0.5 Ejector cylinder is back
       AND     Em_Stop      'S10 to I0.15 Emergency stop (breaker)
       AND N   Stop         'S4 to I0.9 Stop switch (breaker)
       AND     Start        'S1 to I0.6 Start button
 THEN  RESET   Aminus       'Y2 to O0.1 Clamp cylinder back
       SET     Aplus        'Y1 to O0.0 Clamp the workpiece

STEP
 IF            Released     'B2 to I0.1 Clamp cylinder is forward
 THEN  SET     Driller      'Y3 to O0.2 Drill cylinder

STEP
 IF            Drill_dow    'B3 to I0.2 Drill clinder is down
 THEN  RESET   Driller      'Y3 to O0.2 Drill cylinder

STEP
 IF            Drill_up     'B4 to I0.3 Drill clinder is up
 THEN  RESET   Aplus        'Y1 to O0.0 Clamp the workpiece
       SET     Aminus       'Y2 to O0.1 Clamp cylinder is back

STEP
 IF            Released     'B1 to I0.0 Clamp cylinder is released
 THEN  SET     Ejector      'Y4 to O0.3 Ejector

STEP
 IF            Ejected      'B5 to I0.4 Ejector cylinder is forward
 THEN  RESET   Ejector      'Y4 to O0.3 Ejector
       JMP TO Aplus
```

Fig. 5/3

Only the step Aplus is ’jumped to’ and thus requires a step name. Step names may

- use at most 9 characters
- not have any special characters or any blank spaces (both ’A+’ and ’A plus’ are prohibited)
- never be used twice (within a program)
- also have numbers (’Step1’ is a valid name, as is ’1’).

## 5.5 Jumping from step to step

The condition to go the next step allows you to switch from one step to the next. This is basically an implicit jump, that is, a jump that is programmed without it being expressly written. You could also write the jumps without making the program any better or worse; it would simply mean a bit more work.

```
"" Automatic mode of the drilling machine

STEP Aplus
 IF              Released    'B1 to I0.0 Clamp cylinder is released
       AND       Drill_up    'B4 to I0.3 Drill clinder is up
       AND       Ej_back     'B6 to I0.5 Ejector cylinder is back
       AND       Em_Stop     'S10 to I0.15 Emergency stop (breaker)
       AND N     Stop        'S4 to I0.9 Stop switch (breaker)
       AND       Start       'S1 to I0.6 Start button
 THEN  RESET     Aminus      'Y2 to O0.1 Clamp cylinder back
       SET       Aplus       'Y1 to O0.0 Clamp the workpiece
       JMP TO Bplus

STEP Bplus
 IF              Released    'B2 to I0.1 Clamp cylinder is forward
 THEN  SET       Driller     'Y3 to O0.2 Drill cylinder
       JMP TO Bminus

STEP Bminus
 IF              Drill_dow   'B3 to I0.2 Drill clinder is down
 THEN  RESET     Driller     'Y3 to O0.2 Drill cylinder
       JMP TO Aminus

STEP Aminus
 IF              Drill_up    'B4 to I0.3 Drill clinder is up
 THEN  RESET     Aplus       'Y1 to O0.0 Clamp the workpiece
       SET       Aminus      'Y2 to O0.1 Clamp cylinder is back
       JMP TO Cplus

STEP Cplus
 IF              Released    'B1 to I0.0 Clamp cylinder is released
 THEN  SET       Ejector     'Y4 to O0.3 Ejector
       JMP TO Cminus

STEP Cminus
 IF              Ejected     'B5 to I0.4 Ejector cylinder is forward
 THEN  RESET     Ejector     'Y4 to O0.3 Ejector
       JMP TO Aplus
```

Fig. 5/4

However, there are hardly any machines anymore that work in such a simple and straightforward manner as attempted by this very first drilling machine program. We will usually come across branching (and will also be expanding the drilling machine program).

Take the following example:

```
STEP Example
IF                      Large
 THEN   JMP TO          Large pack

IF                      Small
 THEN   JMP TO          Small pack

IF              N       Large
        AND     N       Small
 THEN   JMP TO Error
```

Fig. 5/5

In this step the decision is taken on how many packaging sizes should be produced – either large or small. If the packaging size is not defined, then something is wrong; consequently, the program jumps to the Error step.

What will happen now if due to a strange situation both large and small packages should be produced at the same time, that is, if the variables Large and Small are both true (’1’ or ’High’ or ’TRUE’)?

Then FST works through the program in strict adherence to the line order:
The size is asked for in the first line. If this condition is met, the program will jump to the step Large pack. Then it continues in this step. The query for Small will therefore no longer be read at all.

General:
If the condition is true, a jump is made immediately in the event of an IF ... THEN JMP TO. All the program lines are no longer processed after the jump.

## 5.6 The last step

There must be a jump in the last step of a program. Otherwise the program will be switched off as soon as the last condition has been met.

Most programmers add an unconditional jump. That would look like this in our example:

```
STEP Cminus
 IF              Ejected     'B5 to I0.4 Ejector cylinder is forward
 THEN   RESET    Ejector     'Y4 to O0.3 Ejector
        JMP TO Aplus
 IF                NOP
 THEN   JMP TO Cminus
```

Fig. 5/6

The IF NOP is always true, being an unconditional condition. Consequently the same step continues to be worked on in this last step until the condition IF Ejected has been met.

## 5.7 The alternative sequencer

It is necessary in nearly all projects to make decisions mutually excluding one another:

– Is Stop actuated or not?

– Is Version 1 or Version 2 being produced?

– Is there an error?

The query about the alternative is inserted for the programming in a step.

You already saw an example of this in section 5.5:

```
STEP Example
 IF                    Large
 THEN   JMP TO Large pack

 IF                    Small
  THEN  JMP TO Small pack

 IF             N      Large
       AND      N      Small
  THEN  JMP TO Error
```

Fig. 5/7

**Please note**
The active step (in this case the STEP Example) becomes inactive as soon a condition is met and a jump is made. A clear priority rule is thus present if multiple conditions are met.

If both the conditions 'Large' and 'Small' are met in the example above, then a jump is made to the STEP Large pack.

## 5.8 The parallel sequencer

Multiple sequences must be processed in parallel in many similar cases. The FST statement list does not have a parallel sequencer in any programs. If two sequencers should be processed in parallel, a second program containing the parallel sequencer is simply activated.

Flags can be used to synchronise the sequencers.

# Multitasking with FST: The drilling machine's operating modes

## Chapter 6

# Contents

We have previously used only two programs for the drilling machine, to be precise only one, as the program P1 is started only in the program P0. However, using programs can bring us a great deal further. So here are some explanations about the programs' concept.

## 6.1 Programs work simultaneously

FST supports up to 64 programs, all of which can work simultaneously. 'Simultaneously' means that they appear to work at the same time. However, that is not completely true, of course. For the central unit does not have 64 microprocessors working through the 64 programs, rather, just one. As a consequence, this one processor switches very quickly from one program to another. A complete loop of the processor, a processor cycle, thus processes all 64 programs one after the other.

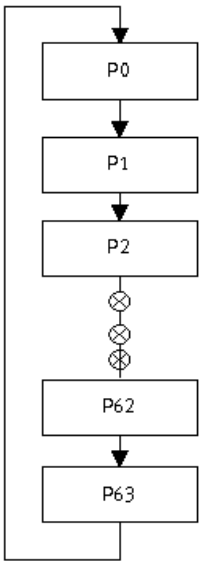



Fig. 6/1

And then there are the inputs and outputs at the beginning and end of processing: The signal statuses are read at the beginning and written at the end.

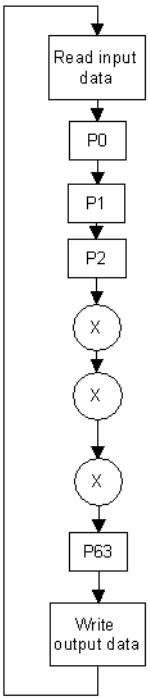



Fig. 6/2

The processor stops for a brief moment to check and see if there is something to be done in the program; if yes, it takes care of it there and moves on to the next program. This happens so quickly that it appears on the outside that all the programs are running at the same time – simultaneouly, or 'multitasking'.

## 6.2 Supervising programs

We have already pointed out that only the program P0 plays a special role; it is automatically activated when the controller is switched on. All the other programs never activate on their own; rather, they must be explicitly activated in the project. As a consequence, operations are needed to supervise programs. There are two operations in particular for this:
P and PS.

**P** is the program itself. A program can be switched on (SET) and off (RESET). A program can also be queried as to whether it is active using IF P5 or IF N P5, just as an operand can be used for a contact or a coil in the ladder diagram.

**PS** is the program status. Each program also has a program status. For example, PS0 belongs to P0. The program status can halt a program during program execution (RESET PS5) or it can continue execution (SET PS5). You can also ask if the program status is active (IF PS5) or inactive (IF N PS5).

What is the difference between program and program status?

– The program status PS is completely meaningless for the ladder diagram. It exists but is not needed.

– In statement list the program status is important in processing programs with the STEP operation.

Take the automatic program for the drilling machine. If for instance the program is in the Bplus step, the driller should be run down.

For some reason (Emergency Stop, material jam or something else), the machine is halted, the driller is changed, the material is taken out and the process is supposed to start from the beginning again. For this purpose the Program P1 is first reset (RESET P1) and then restarted from the beginning (SET P1).

The program has to be halted for another reason (RESET PS1). If the program is continued using SET PS1, it will continue working in the step Bplus and will not start over from the beginning.

Program and program status can be summarised as follows:

| **Program P** | **Program status PS** | **Status** |
|---|---|---|
| 0 | 0 | inactive |
| 0 | 1 | reserved |
| 1 | 0 | active but halted |
| 1 | 1 | active and processed |

Tab. 6/1

## 6.3 Starting programs with time limits

As described, the programs are processed simultaneously. Cycle times depend on the length of the program. The more program lines you pack into a program, the longer the cycle time will be. However, there are applications which should not be continuously processed but perhaps one every second or minute. You can use the module F4 to make this happen. Here it is agreed upon that a program will be processed with time limits, whereas the order of processing – from P0 to P63 and back to P0 – does not change. Should the processor come upon a program called up with time limits, then it will check whether the desired time is elapsed; if yes, this program is processed, if not, it is passed over.

## 6.4 Exchanging data among programs

Inputs, outputs, flags, registers, times and counters are global in FST; i.e., they apply to all the programs. Communication among the programs occurs in the usual way via flags. There are up 160 000 individual ones which can be addressed in 10 000 flag words, thus from F0.0 to F9999.15.

# The drilling machine grows in statement list

## Chapter 7

# Contents

The long preface merely served the purpose of bringing the drilling machine program somewhat closer to the reality of modern machine controllers. After all, the organisation of operating modes such as automatic, manual, etc. must be programmed first.

## 7.1 Starting and stopping the automatic program

The automatic program is started using 'Start' and can then work cyclically (a query comes later on whether material is still available in the drop magazine).
However, it may only be started if the machine is in home position, neither stop, halt nor emergency stop is actuated and the back to home position program and the manual program are not active.

We can therefore program as follows:

```
"" Switch on automatic mode
 IF            Start       'S1: Start button
"" Home position
       AND     Released    'B1 to I0.0 Clamp cylinder is released
       AND     Drill_up    'B4 to I0.3 Drill clinder is up
       AND     Ej_back     'B6 to I0.5 Ejector cylinder is back
"" Halt buttons
       AND     Halt        'S2 to I0.7 Halt button (breaker)
       AND     Stop        'S4 to I0.9 TStop switch (breaker)
       AND     Em_Stop     'S10 to I0.15 Emergency stop (breaker)
""Programs
       AND  N  P2          'Back to home position
       AND  N  P4          'Manual mode
 THEN  SET     P1          'Automatic and inching
```

Fig. 7/1

As the automatic mode should now run in continous mode, the automatic program must be modified.

```
"" Automatic mode of the drilling machine

STEP Aplus
 IF              Released     'B1 to I0.0 Clamp cylinder is released
       AND       Drill_up     'B4 to I0.3 Drill clinder is up
       AND       Ej_back      'B6 to I0.5 Ejector cylinder is back
 THEN  RESET     Aminus       'Y2 to O0.1 Clamp cylinder back
       SET       Aplus        'Y1 to O0.0 Clamp the workpiece

STEP Bplus
 IF              Clamped      'B2 to I0.1 Clamp cylinder is forward
 THEN  SET       Driller      'Y3 to O0.2 Drill cylinder

STEP Bminus
 IF              Drill_dow    'B3 to I0.2 Drill clinder is down
 THEN  RESET     Driller      'Y3 to O0.2 Drill cylinder

STEP Aminus
 IF              Drill_up     'B4 to I0.3 Drill clinder is up
 THEN  RESET     Aplus        'Y1 to O0.0 Clamp the workpiece
       SET       Aminus       'Y2 to O0.1 Clamp cylinder is back

STEP Cplus
 IF              Released     'B1 to I0.0 Clamp cylinder is released
 THEN  SET       Ejector      'Y4 to O0.3 Ejector

STEP Cminus
 IF              Ejected      'B5 to I0.4 Ejector cylinder is forward
 THEN  RESET     Ejector      'Y4 to O0.3 Ejector
       JMP TO Aplus
 IF              NOP
 THEN  JMP TO Cminus
```

Fig. 7/2

Although the automatic program runs endlessly, it cannot be halted any more.
Two buttons are provided for this in the operation: STOP – stops immediately, HALT – stops after the end of the cycle.

STOP is very easy to program; it merely resets the automatic program.

HALT is a bit more complicated, as the HALT button can be actuated anytime during the program cycle but only becomes effective at the end of the cycle. Therefore, we will need a flag to note that HALT has been actuated. This flag must then intervene at the end of the cycle.

The organisation could be expanded in this way:

```
"" Stop automatic
 IF            N       Stop      'S4 to I0.9 Stop button (breaker)
 THEN   RESET          P1        'Automatic and inching""

Halt after end of cycle
 IF            N       Halt      'S2 to I0.7 Halt button (breaker)
 THEN   SET            F_HALT    'Flag HALT actuated
```

Fig. 7/3

7. The drilling machine grows in statement list

We can expand the last step in automatic mode:

```
STEP Cminus
 IF              Ejected     'B5 to I0.4 Ejector cylinder is forward
 THEN   RESET    Ejector     'Y4 to O0.3 Ejector

STEP Halt
 IF              Ej_back     'B6 to I0.5 Ejector cylinder is back
       AND  N    F_HALT      'Flag HALT actuated
 THEN  JMP TO Aplus

IF               F_HALT      'Flag HALT actuated
 THEN  RESET     F_HALT      'Flag HALT actuated
       RESET     P1          'Automatic and inching

IF               NOP
 THEN  JMP TO Halt
```

Fig. 7/4

If a minimal program is added to the programs P2 and P4 now being used, we can thus compile and use the modified project.

```
IF      NOP
THEN    NOP
```

Fig. 7/5

## 7.2 Inching mode

Inching mode means that the automatic sequence is processed in steps, whereas going from one step to the next depends on additional input from operating personnel. An inching push button is required which in our example is implemented with the start push button into practice.

You could formulate this in general as follows:

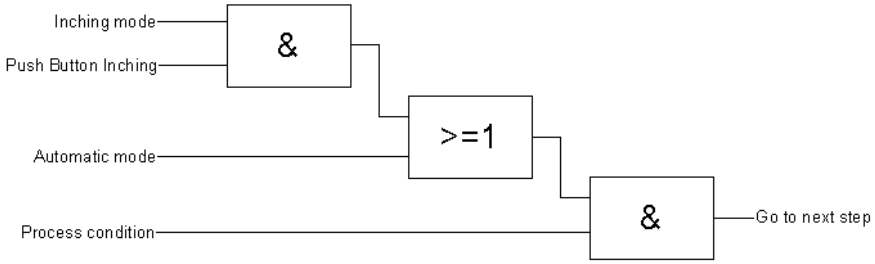



Fig. 7/6

Now this condition could be worked into every step, of course. Let's take the step Aminus for example.

```
STEP Aminus
 IF               Drill_up      'B4 to I0.3 Drill clinder is up
 THEN   RESET     Aplus         'Y1 to O0.0 Clamp the workpiece
        SET       Aminus        'Y2 to O0.1 Clamp cylinder is back
```

Fig. 7/7

If the condition to go the next step should be represented as it was previously, we must program as follows:

7. The drilling machine grows in statement list

```
STEP Aminus
 IF             Drill_up      'B4 to I0.3 Drill clinder is up
       AND   (  Continuous    'S3 to I0.8 Continuous (0=Inching mode)
       OR    N  Continuous    'S3 to I0.8 Continuous (0=Inching mode)
       AND      Start  )      'S1 to I0.6 Start button
 THEN  RESET    Aplus         'Y1 to O0.0 Clamp the workpiece
       SET      Aminus
```

Fig. 7/8

This procedure would be possible; but it inflates the automatic program unnecessarily. The combined AND/OR link, which influences going to the next step independently from the process condition, can be programmed in the organisation program and used as a flag in automatic mode. We can program for once in the organisation program:

```
"" Forward flag for inching mode
 IF     P1              'Automatic and inching
        AND   (  Continuous    'S3 to I0.8 Continuous (0=Inching mode)
        OR    N  Continuous    'S3 to I0.8 Continuous (0=Inching mode)
        AND      Start      )  'S1 to I0.6 Start button
 THEN   SET      F_forward     'Flag advance condition inching
 OTHRW  RESET    M_Continue    'Flag advance condition inching
```

Fig. 7/9

To be able to insert this flag into the automatic program, which must wait in inching mode:

```
STEP Aminus
 IF               Drill_up     'B4 to I0.3 Drill clinder is up
        AND       F_Forward    'Flag go to next step condition inching
 THEN   RESET     Aplus        'Y1 to O0.0 Clamp the workpiece
        SET       Aminus       'Y2 to O0.1 Clamp cylinder is back
```

Fig. 7/10

The program [21)] could now look like this:

```
"" Automatic mode of the drilling machine

STEP Aplus
 IF            Released    'B1 to I0.0 Clamp cylinder is released
       AND     Drill_up    'B4 to I0.3 Drill clinder is up
       AND     Ej_back     'B6 to I0.5 Ejector cylinder is back
 THEN  RESET   Aminus      'Y2 to O0.1 Clamp cylinder back
       SET     Aplus       'Y1 to O0.0 Clamp the workpiece

STEP Bplus
 IF            Clamped     'B2 to I0.1 Clamp cylinder is forward
       AND     F_Forward   'Flag go to next step condition inching
 THEN  SET     Driller     'Y3 to O0.2 Drill cylinder

STEP Bminus
 IF            Drill_dow   'B3 to I0.2 Drill clinder is down
       AND     F_Forward   'Flag go to next step condition inching
 THEN  RESET   Driller     'Y3 to O0.2 Drill cylinder

STEP Aminus
 IF            Drill_up    'B4 to I0.3 Drill clinder is up
       AND     F_Forward   'Flag go to next step condition inching
 THEN  RESET   Aplus       'Y1 to O0.0 Clamp the workpiece
       SET     Aminus      'Y2 to O0.1 Clamp cylinder is back

STEP Cplus
 IF            Released    'B1 to I0.0 Clamp cylinder is released
       AND     F_Forward   'Flag go to next step condition inching
 THEN  SET     Ejector     'Y4 to O0.3 Ejector

STEP Cminus
 IF            Ejected     'B5 to I0.4 Ejector cylinder is forward
       AND     F_Forward   'Flag go to next step condition inching
 THEN  RESET   Ejector     'Y4 to O0.3 Ejector
       JMP TO Aplus
 IF            NOP
 THEN  JMP TO Cminus
```

Fig. 7/11

21 You will find the project as Drill_03.ZIP on your CD.

## 7.3 Edge detection

The inching mode, programmed in this way, now has a characteristic viewed as being either 'correct' or 'wrong', depending on the application.

If the machine is in inching mode, it is sufficient to rest on the Start push button (keep it held down) to ensure that the sequence is continuously processed.
It is demanded in some applications that the machine can work exactly one step forward when a push button is actuated.

What we really want – in this case – is for the machine to go to the next step when your finger starts to actuate the Start push button. The term for this in automation technology is edge detection.

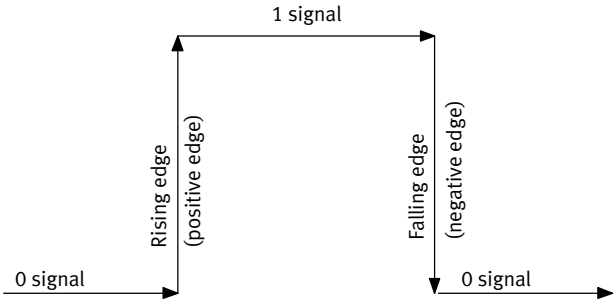



Fig. 7/12

Going from step to step should take place by means of the rising edge but not the 1 signal.

## 7.3.1 Programming for edge detection

Edge detection is programmed very simply in FST using a flag.

```
"Edge detection
 IF           N   Edge_flag    'Flag edge detection Start button
       AND       Start        'S1 to I0.6 Start button
 THEN  SET       Edge_flag    'Flag edge detection push button start
 IF           N   Start        'S1 to I0.6 Start button
 THEN  RESET     Edge_flag    'Flag edge detection Start button
```

Fig. 7/13

The condition 'IF N Edge_flag AND Start' is met for exactly one controller cycle if the Start button is actuated.

You get a negative edge when you reverse the logic.

```
"" Negative edge
 IF           N    Edge_n     'Flag for negative edge detection
       AND    N    Start      'S1 to I0.6 Start button
 THEN  SET         Edge_n     'Flag for negative edge detection
 IF                Start      'S1 to I0.6 Start button
 THEN  RESET       Edge_n     'Flag for negative detection
```

Fig. 7/14

If we want to have the inching mode for the drilling machine programmed in such a way that the Start button has to be actuated for each step, then we can expand our organisation program:

```
""Organisation program for the drilling machine
"" Switch on automatic mode
 IF            Start        'S1: Start button
"" Home position
      AND      Released     'B1: Clamp cylinder is released
      AND      Drill_up     'B4: Drill cyllinder is up
      AND      Ej_back      'B6: Ejector cylinder is back
"" Halt buttons
      AND      Halt         'S2: Halt button (breaker)
      AND      Stop         'S4: Stop button (breaker)
      AND      Em_Stop      'S10: Emergency stop switch (breaker)
"" Programs
      AND   N  P2           'Back to home position
      AND   N  P4           'Manual mode
 THEN SET      P1           'Automatic program

"" Stop automatic
 IF         N  Stop         'S4: Stop button (breaker)
 THEN RESET    P1           'Automatic program

"" Halt after end of cycle
 IF         N  Halt         'S2: Halt button (breaker)
 THEN SET      F_HALT       'Flag HALT actuated

"" Reset flag for inching mode
 IF            NOP
 THEN SET      F_forward    'Flag advance condition inching

"" Edge detection and set flag for inching mode
 IF         N  Edge_flag    'Flag edge detection Start button
      AND      Start        'S1: Start button
      AND      P1           'Automatic program
      AND   (  Continuous   'S3: Contin. mode switch (0=inching mode)
      OR    N  Continuous ) 'S3: Contin. mode switch (0=inching mode)
 THEN SET      Edge_flag    'Flag edge detection push button start
      SET      F_Forward    'Flag go to next step condition inching
 IF         N  Start        'S1: Start button
 THEN RESET    Edge_flag    'Flag edge detection push button start
```

Fig. 7/15

## 7.4 The home position program

The home position program[22] is quickly programmed for such a small and simple system. The home position program can be very complex for larger systems, as it should bring the machine to home position from any other position.

In our case, you can simply program as follows:
C- B- A-

P2 will look somewhat like this as a result:

```
"" Run ejector to home position
STEP
 IF            NOP
 THEN  RESET   Ejector      'O0.3: Y3: Ejector

"" Run drill to home position
STEP
 IF            Ej_back      'I0.5: B6: Ejector cylinder is back
 THEN  RESET   Driller      'O0.2: Y3: Drill cylinder

"" Run clamp cylinder to home position
STEP
 IF            Drill_up     'I0.3: B4: Drill cyllinder is up
 THEN  RESET   Aplus        'O0.0: Y1: Clamp the workpiece
       SET     Aminus       'O0.1: Y2: Release the workpiece

"" End home position program
STEP
 IF            Released     'I0.0: B1: Clamp cylinder is released
 THEN  RESET   P2           'Back to home position
```

Fig. 7/16

Please observe that the home position program automatically switches off in this example. Once the home position has been achieved, program P2 is switched off.

22) You will find the project including the home position program as Drill_05.ZIP on your CD.

Depending on the operating philosophy, it must be clarified under which conditions the home position program is called up in the first place. You certainly would not want to allow the home position button to react in the middle of an automatic sequence. It might suffice for the purpose of this example to ask whether the automatic program has been stopped (not halted). The organisation program is expanded for this:

```
"" Back to home position
 IF         N   P1         'Automatic program
      AND       Em_Stop    'I0.15: S10: Emergency stop switch (breaker)
      AND       Home_pos   'I0.11: S6: Back to home position button
 THEN  SET      P2         'Back to home position
```

Fig. 7/17

Finally, we installed a light which should indicate if the home position program is active. This is very easy to program in the organisation program.

```
IF             P2          'Back to home position
 THEN   SET    L_Home      'O0.5: H2 Back to home position light
 OTHRW  RESET  L_Home      'O0.5: H2 Back to home position light
IF             P1          'Automatic program
 THEN   SET    L_Auto      'O0.7: H4: Automatic continuous mode light
 OTHRW  RESET  L_Auto      'O0.7: H4: Automatic continuous mode light
```

Fig. 7/18

# The drilling machine grows in ladder diagram

## Chapter 8

# Contents

The long preface merely served the purpose of bringing the drilling machine program somewhat closer to the reality of modern machine controllers. After all, the organisation of operating modes such as automatic, manual, etc. must be programmed first.

## 8.1 Starting and stopping the automatic program

The automatic program is started using 'Start' and can then work cyclically (a query comes later on whether material is still available in the drop magazine).
However, it may only be started if the machine is in home position, neither stop, halt nor emergency stop is actuated and the back to home position program and the manual program are not active.

We can therefore program as follows:

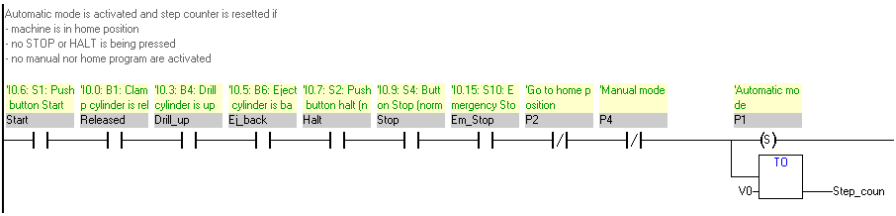



Fig. 8/1

As the automatic mode should now run in continuous mode, the start button must be deleted in the first step in the automatic program.

Although the automatic program runs endlessly, it cannot be halted any more. Two buttons are provided for this in the operation:

– STOP stops immediately,

– HALT stops after the end of the cycle.

STOP is very easy to program; it merely resets the automatic program.

HALT is a bit more complicated, as the HALT button can be actuated anytime during the program cycle but only becomes effective at the end of the cycle. Therefore, we will need a flag to note that HALT has been actuated. This flag must then intervene at the end of the cycle.

The organisation could be expanded in this way:

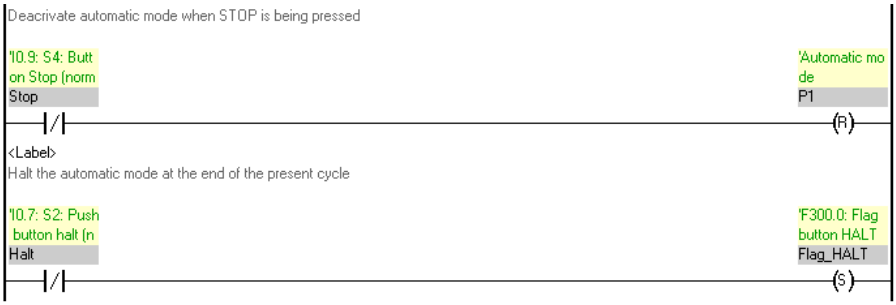



Fig. 8/2

We can expand the last step in automatic mode:

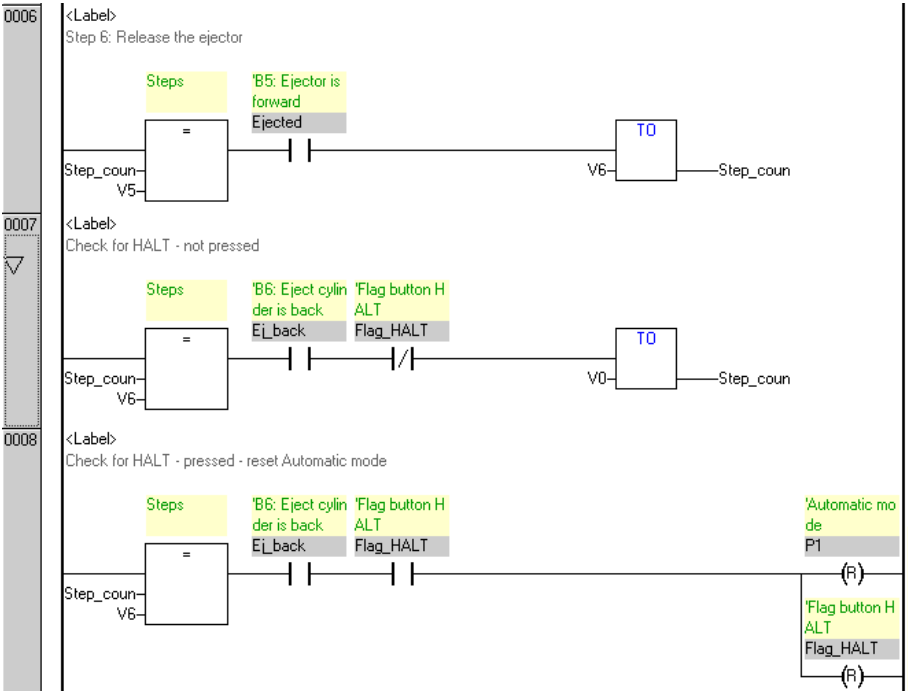



Fig. 8/3

If a minimal program is added to the programs P2 and P4 now being used, we can thus compile and use the modified project [23].

[23] You will find this status as Drill_L3.ZIP on your CD.

## 8.2 Inching mode

Inching mode means that the automatic sequence is processed in steps, the advancement from one step to the next depending on additional input from operating personnel. An inching push button is required which in our example is implemented with the start push button into practice.

You could formulate this in general as follows:

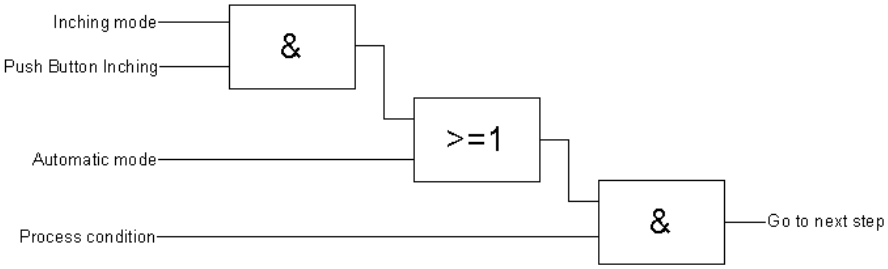



Fig. 8/4

Now this condition could be worked into every step, of course. However, that would be quite time-consuming and would reduce clarity. The combined AND/OR link, which influences going to the next step independently from the process condition, can be programmed in the organisation program and used as a flag in automatic mode.

We can program the following in the organisation program:

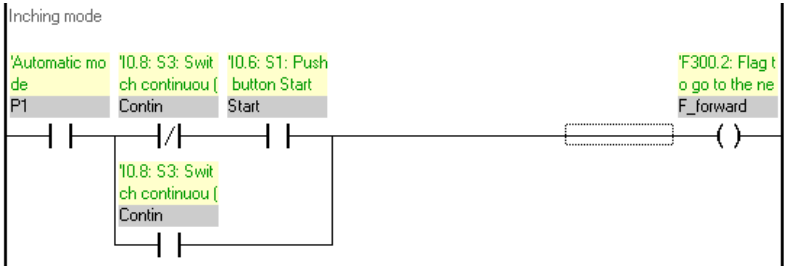



Fig. 8/5

to be able to insert this flag into the automatic program in each step which must wait in inching mode:
As an example [24] a step would look like this:

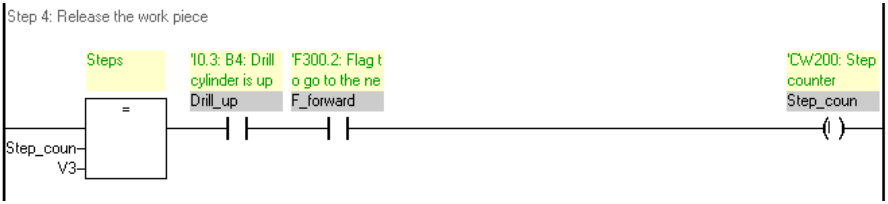



Fig. 8/6

[24] You will find the entire project with this supplement as Drill_L4.ZIP on your CD.

## 8.3 Edge detection

The inching mode, programmed in this way, now has a characteristic viewed as being either 'correct' or 'wrong', depending on the application.

If the machine is in inching mode, it is sufficient to rest on the Start push button (keep it held down) to ensure that the sequence is continuously processed. It is demanded in some applications that the machine can work exactly one step forward when a push button is actuated.

What we really want – in this case – is for the machine to go to the next step when your finger starts to actuate the Start push button. The term for this in automation technology is edge detection.

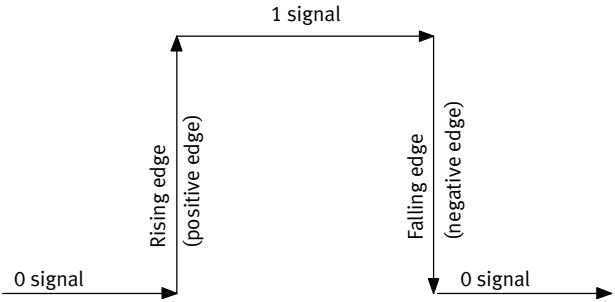



Fig. 8/7

Going from step to step should take place by means of the rising edge but not the 1 signal.

## 8.3.1 Programming for edge detection

Edge detection is programmed very simply using a flag.

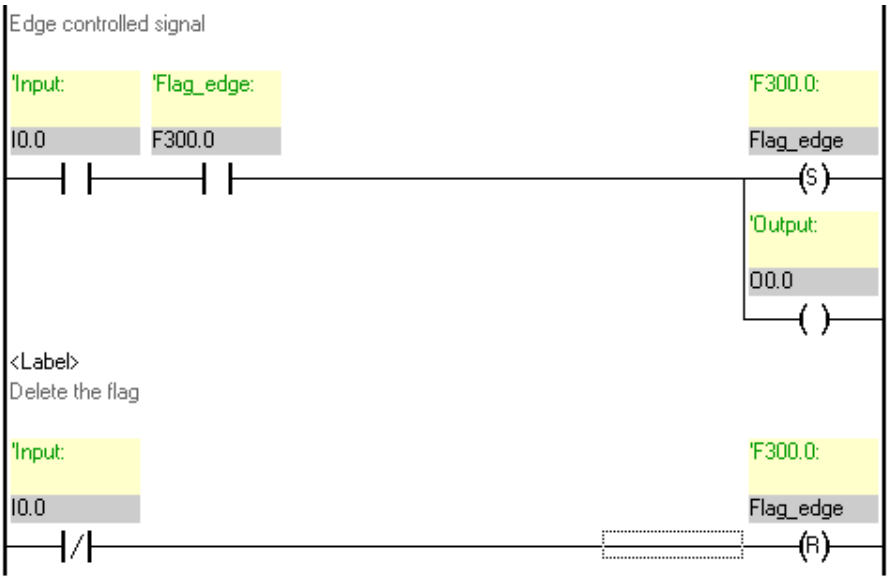



Fig. 8/8

In this program the positive edge is detected and switches the output O0.0 to 1 for exactly one cycle. You get a negative edge when you reverse the logic.

If we want to have the inching mode for the drilling machine programmed [25] in such a way that the Start button has to be actuated for each step, then we can expand our organisation program:

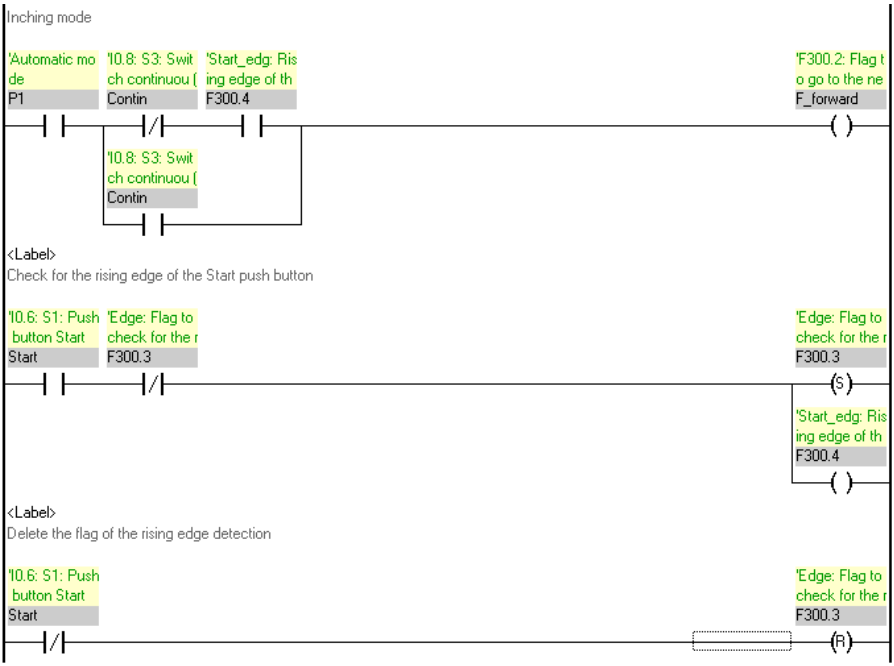



Fig. 8/9

[25] You will find the project as Drill_L5.ZIP on your CD.

8. The drilling machine grows in ladder diagram

## 8.4 The home position program

The home position program [26] is quickly programmed for such a small and simple system. The home position program can be very complex for larger systems, as it should bring the machine to home position from any other position.

In our case, you can simply program as follows:
C- B- A-

P2 will look somewhat like this as a result:

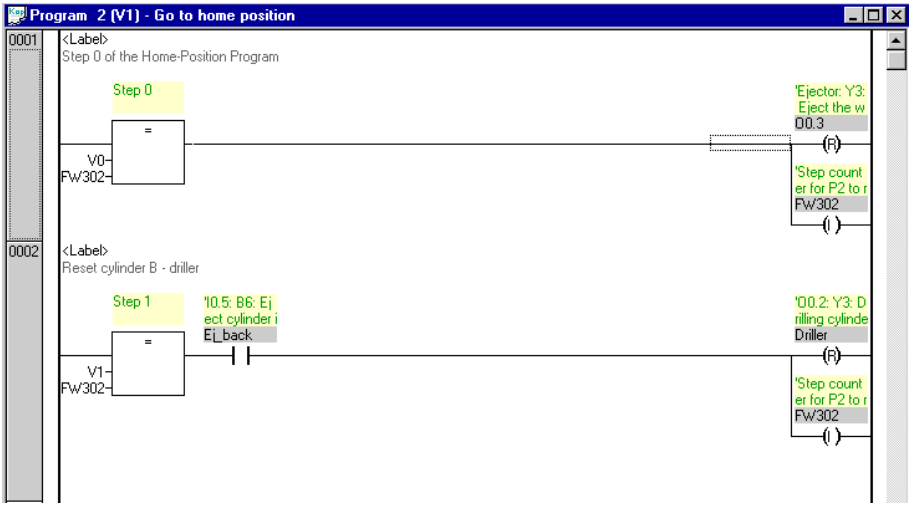



Fig. 8/10

[26] You will find the project including the home position program as Drill_L6.ZIP on your CD.

8. The drilling machine grows in ladder diagram

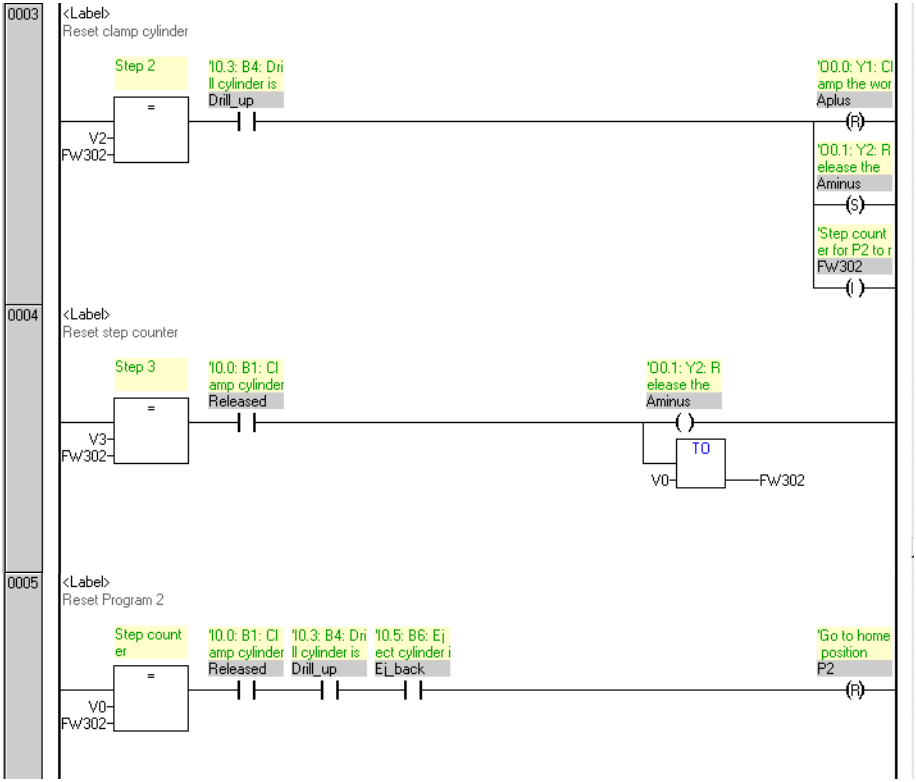



Fig. 8/11

Please observe that the home position program automatically switches off in this example. Once the home position has been achieved, program P2 is switched off.

Depending on the operating philosophy, it must be clarified under which conditions the home position program is called up in the first place. You certainly would not want to allow the home position button to react in the middle of an automatic sequence. It might suffice for the purpose of this example to ask whether the automatic program has been stopped (not halted). The organisation program is expanded for this:


Fig. 8/12

Finally, we installed a light which should indicate if the home position program is active. This is programmable as an example in the orgranisation program.

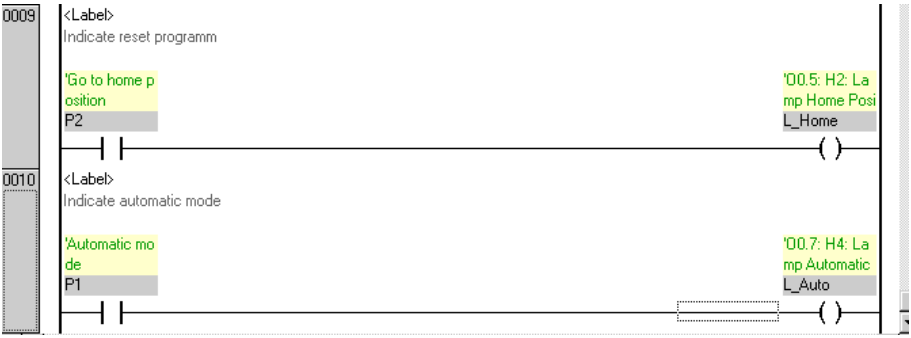



Fig. 8/13

8. The drilling machine grows in ladder diagram

# Times and counters with FST in statement list

## Chapter 9

# Contents

There will very unlikely be any automation projects in which times are not used and only a few in which counters are not used. For that reason we should deal with times and counters in detail.

Times and counters belong together, as a time module is a counter that counts clock pulses. However, the time module is somewhat easier to manage than a counter because some counter functions are hidden. Let's start with the times.

## 9.1 The time module [27)]

A time module can be represented as seen below:

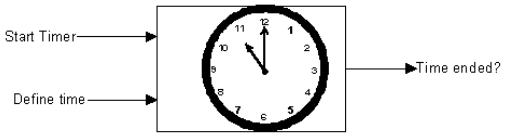



Fig. 9/1

A time module must have at least three connections:

– an input for the start

– an input to set the time

– an output that reports whether or not the module is active.

In addition, it is possible to display the elapsed time.

[27)] You will find all the time examples on the CD under Time_en.ZIP

These inputs and outputs receive a name in FST:

- The time module (timer) itself is called T. As there are 256 timers, you thus have the timers T0 ... T255
- The time is started by setting this module: SET T0
- The set time, the nominal value, is in the timer preselect, TP0 ... TP255
- The time module itself becomes ’1’ after being called up and ’0’ after the time has elapsed
- The current time value, the actual value, is in the timer words, TW0 ... TW255

## 9.1.1 Let’s begin with a simple example

When the input I0.6 on our control is actuated, the output O0.6 should be active for a period of 5 seconds:

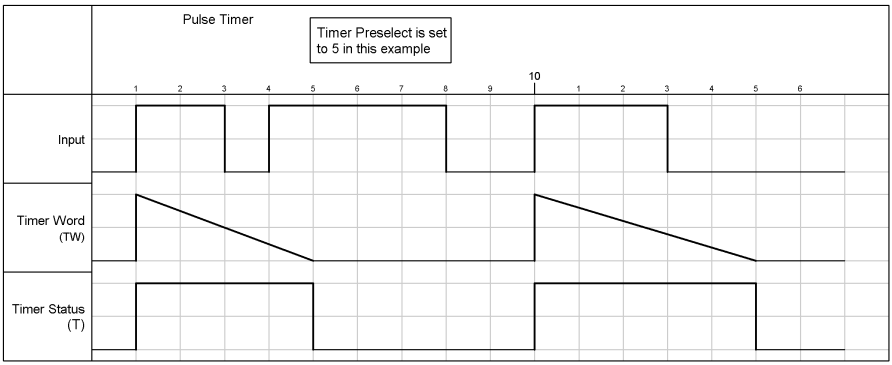



Fig. 9/2

The program can read as follows:

```
STEP Start
 IF                  I0.6      'Input to start the timer
 THEN   SET          T0        'Pulse timer
        WITH         5s

STEP Time_eval
 IF                  T0        'Pulse timer
 THEN   SET          O0.6      'Input showing pulse
 OTHRW  RESET        O0.6      'Output showing pulse

 IF            N     T0        'Pulse timer
        AND    N     I0.6      'Input to start the timer
 THEN   JMP TO Start

 IF                  NOP
 THEN   JMP TO Time_eval
```

Fig. 9/3

Let's take a closer look at the program:

```
THEN    SET     T0
```

Fig. 9/4

switches on the time module. The time runs, the timer status T0 is active ('1').

```
WITH            5s
```

Fig. 9/5

defines the time. Here permissible values are 0.01 ... 655.35 s. Please observe the decimal point (not a decimal comma).

In the online display we see the status, the current value (actual value) and the preselect value (nominal value).

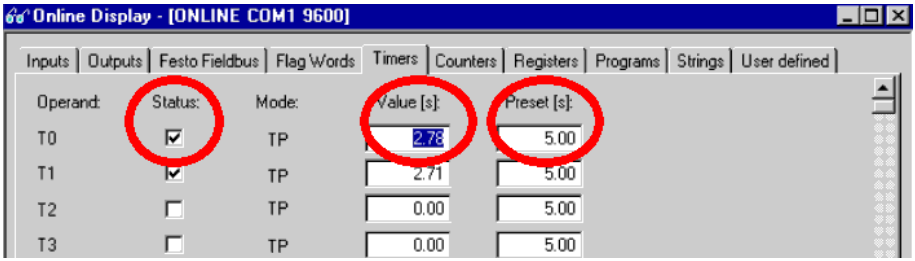



Fig. 9/6

The query

```
IF      T0      'Pulse timer
```

Fig. 9/7

queries the Boolean status of the time module (timer). If the timer is '1', then the timer is still active and the time is not yet elapsed. If the status is '0', then the time is elapsed and the timer has done its work and is ended (inactive).

## 9.1.2 The timer without steps

SET T0 restarts the timer. In the sequencer used above as an example, the step-by-step sequence ensures that the timer is not restarted until the time is ended or the input I0.6 has become '0' after the time has elapsed.

If you are not using a sequencer, the timer is locked:

```
IF                I0.6     'Input to start the timer
      AND     N   T1       'Pulse timer
THEN  SET         T1       'Pulse timer
      WITH        5s

IF                T1       'Pulse timer
THEN  SET         O0.7     'Display pulse timer without steps
OTHRW RESET       O0.7     'Display pulse timer without steps
```

Fig. 9/8

The usual time modules should then be programmed.

## 9.1.3 The timer ON delay

The timer ON delay is characterised by the output becoming 1 when the time is over AND the input is still 1. Here is a representation of this:

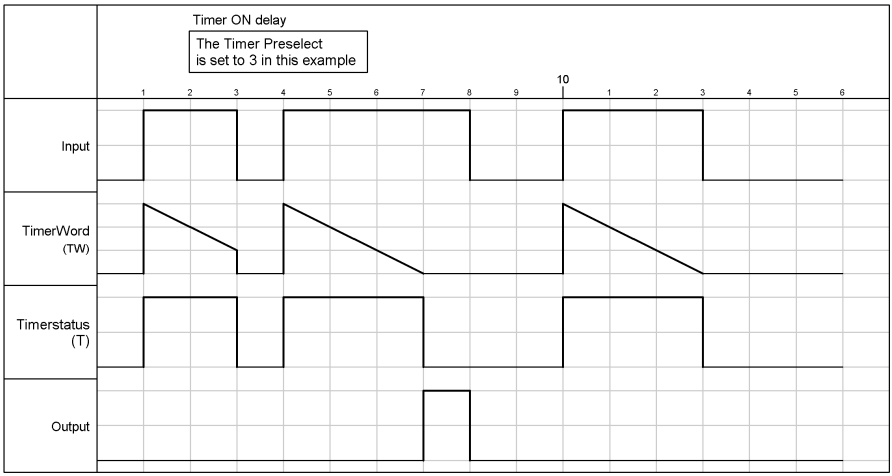



Fig. 9/9

The timer ON or switch ON delay can be programmed with or without steps.

First an example with steps:

```
STEP Start
"" The time starts with input I0.5
 IF                I0.5      'Input for timer ON delay
 THEN   SET        T2        'Timer ON delay with steps
        WITH       5s
STEP Wait
"" If the input becomes 0, the time is canceled
 IF            N   I0.5      'Input for timer ON delay
 THEN   RESET      T2        'Timer ON delay with steps
        JMP TO Start
"" If the time has elapsed and the input is still active, the
"" Output is switched on
 IF            N   T2        'Timer ON delay with steps
        AND        I0.5      'Input for timer ON delay
 THEN   SET        O0.5      'Output timer ON delay steps
STEP End
"" If the input becomes 0, the output is switched off
 IF            N   I0.5      'Input for timer ON delay
 THEN   RESET      O0.5      'Output timer ON delay steps
        JMP TO Start
```

Fig. 9/10: Example with steps

```
"" Start the time if it is not already or was started
 IF            N   T3          'Timer ON delay without steps
       AND         I0.5        'Input for timer ON delay
       AND       N F0.0        'Edge flag for timer ON delay
 THEN  SET         T3          'Timer ON delay without steps
       WITH        5s
       SET         F0.0        'Edge flag for timer ON delay
"" Time module and output and edge flag are switched off,
"" as soon as the input becomes 0
 IF            N   I0.5        'Input for timer ON delay
 THEN  RESET       T3          'Timer ON delay without steps
       RESET       O0.4        'Display timer ON delay without steps
       RESET       F0.0        'Edge flag for timer ON delay
"" The output is switched on as soon as the time module is ended
"" and the input is still active
 IF            N   T3          'Timer ON delay without steps
       AND         F0.0        'Edge flag for timer ON delay
       AND         I0.5        'Input for timer ON delay
 THEN  SET         O0.4        'Display timer ON delay without steps
```

Fig. 9/11: Example without steps

## 9.1.4 The switch OFF delay

The switch OFF or timer OFF delay immediately turns on an output with the input and allows it to be turned on for the preselected time after the input has become '0'.

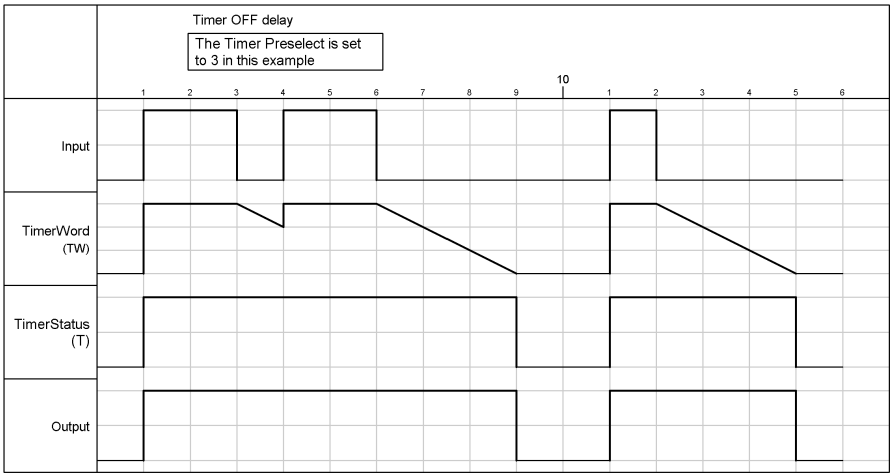



Fig. 9/12

The timer OFF delay with steps can be programmed as follows:

```
STEP Start
 IF                  I0.4     'Start timer OFF delay
 THEN   SET          O0.3     'Timer OFF delay with steps

 IF            N     I0.4     'Start timer OFF delay
       AND           O0.3     'Timer OFF delay with steps
 THEN   SET          T4       'Timer OFF delay with steps
       WITH          5s

STEP Wait
 IF            N     T4       'Timer OFF delay with steps
 THEN   RESET        O0.3     'Timer OFF delay with steps
       JMP TO Start
```

Fig. 9/13

If you are not using steps, you can proceed as follows:

```
"" Switch on output, reset edge flag
 IF                  I0.4       'Start timer OFF delay
 THEN  SET           O0.2       'Timer OFF delay without steps
       RESET         F0.1       'Edge flag timer OFF delay

"" Start time, set edge flag
 IF            N     I0.4       'Start timer OFF delay
       AND           O0.2       'Timer OFF delay without steps
       AND     N     F0.1       'Edge flag timer OFF delay
 THEN  SET           T5         'Timer OFF delay without steps
       WITH          5s
       SET           F0.1       'Edge flag timer OFF delay

"" Time has ended
 IF            N     T5         'Timer OFF delay without steps
       AND           F0.1       'Edge flag timer OFF delay
 THEN  RESET         O0.2       'Timer OFF delay without steps
```

Fig. 9/14

### 9.1.5 The flasher

Let's assume that we want have an output flash with a frequency of 2 seconds (0.5 Hz). Then this output should be '1' for one second and '0' for one second, respectively. Of course, this task can also be programmed with and without steps. First the version with steps.

```
STEP Start
"" Start time of 1 sec. and switch on output
 IF                NOP
 THEN   SET        T6          'Flash time with steps
        WITH       1s
        SET        O0.1        'Flash with steps

STEP Wait
"" If 1 sec. has ended, switch off output and start Off time
 IF            N   T6          'Flash time with steps
 THEN   RESET      O0.1        'Flash with steps
        SET        T6          'Flash time with steps

STEP End
"" After the OFF time has elapsed, jump back
 IF            N   T6          'Flash time with steps
 THEN   JMP TO Start
```

Fig. 9/15

The version without steps can be expressed concisely as follows:

```
"" The timer automatically starts every second
"" The output changes its status every second
 IF            N   T7          'Flash time without steps
 THEN    SET       T7          'Flash time without steps
        WITH       1s
        LOAD   N   O0.0        'Output flashes without steps
          TO       O0.0        'Output flashes without steps
```

Fig. 9/16

## 9.2 The counter module [28)]

The counter can be viewed in the same way that the time module can be represented.
For a comparison with the time module, however, we require additional connections.

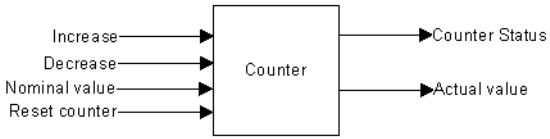



Fig. 9/17

A few practical examples should show the possibilities in handling counters.

### 9.2.1 Count up – increment

Let's assume the following task: The signal at an input should be counted. An output should be turned on after 5 signals. A reset input restores the counter and turns off the output at any time.

28) You will find all the counter module examples on the CD under the title Counter.ZIP

You could program like this in step program:

```
"" Load the set value
STEP Start
 IF                  NOP
 THEN   LOAD         V5
        TO           CP0     'Set value counter 0
        SET          C0      'Increment

"" Count and monitor the reset input
STEP Count
 IF                  I0.1    'Reset counter
 THEN   JMP TO Start

 IF                  I0.0    'Input signal being counted
 THEN   INC          C0      'Increment

"" Evaluate counter
STEP    Forward
 IF                  I0.1    'Reset counter
 THEN   RESET        C0      'Increment
        JMP TO Start

 IF           N      I0.0    'Input signal being counted
        AND          C0      'Increment
 THEN   JMP TO Count

 IF           N      I0.0    'Input signal being counted
        AND     N    C0      'Increment
 THEN   SET          O0.0    'Shows end of counting

"" Reset
STEP    Reset
 IF                  I0.1    'Reset counter
 THEN   RESET        O0.0    'Shows end of counting
        JMP TO Start
 IF                  NOP
 THEN   JMP TO Reset
```

Fig. 9/18

A pure parallel logic program without steps requires an edge flag for the counting input itself as well as a beginning. The flag FI, available in every program, is used here. The initialisation flag is active for exactly one cycle after the program has been switched on.

```
"" Switching on the program
"" or the input I0.1 sets the set value to 5
 IF                FI          'Initialisation flag
       OR          I0.1        'Reset counter
 THEN  LOAD        V5
         TO        CP1         'Nominal value counter 1
       SET         C1          'Counter 1
       RESET       O0.1        'Counter result

"" Count
 IF                I0.0        'Input signal being counted
       AND     N   F0.0        'Edge detection counter
 THEN  SET         F0.0        'Edge detection counter
       INC         C1          'Counter 1

"" Reset edge flag
 IF            N   I0.0        'Input signal being counted
 THEN  RESET       F0.0        'Edge detection counter

"" Evaluate counter
 IF            N   C1          'Counter 1
 THEN  SET         O0.1        'Counter result
```

Fig. 9/19

## 9.2.2 Count backwards – decrement

Decrementing differs only marginally from incrementing. The backwards counter shows how many parts are still 'missing', the forwards counter shows how many parts have already been counted.

After the command 'SET CX', the counter's actual value is loaded into the start value. The nominal value is not viewed any closer, as the target is now 0. The counting procedure itself uses the decrement command, DEC.

Here we have the result with steps.

```
"" Load the set value
STEP Start
 IF                NOP
 THEN   SET        C2          'Decrement with steps
        LOAD       V5
          TO       CW2         'Actual value of counter2

"" Count and monitor the reset input
STEP Count
 IF                I0.3        'Reset decrement
 THEN   JMP TO Start

 IF                I0.2        'Counter input decrement
 THEN   DEC        C2          'Decrement with steps

"" Evaluate counter
STEP Forward
 IF                I0.3        'Reset decrement
 THEN   RESET      C2          'Decrement with steps
        JMP TO Start

 IF           N    I0.2        'Counter input decrement
        AND        C2          'Decrement with steps
 THEN   JMP TO Count

 IF           N    I0.2        'Counter input decrement
        AND     N  C2          'Decrement with steps
 THEN   RESET      O0.2        'Counter result decrement steps

"" Reset
STEP Reset
 IF                I0.3        'Reset decrement
 THEN   RESET      O0.2        'Counter result decrement steps
        JMP TO Start
```

Fig. 9/20

And here is how you could program without steps:

```
"" Switching on the program
"" or the input I0.1 sets the nominal value to 5
 IF                FI        'Initialisation flag
       OR          I0.3      'Reset decrement
 THEN  SET         C2        'Decrement with steps
       LOAD        V5
         TO        CW3       'Actual value counter 3
       RESET       O0.3      'Counter result decrement

"" Count
 IF                I0.2      'Counter input decrement
       AND    N    F0.1      'Edge detection decrement
 THEN  SET         F0.1      'Edge detection decrement
       DEC         C3        'Decrement without steps

"" Reset edge flag
 IF           N    I0.2      'Counter input decrement
 THEN  RESET       F0.1      'Edge detection decrement

"" Evaluate counter
 IF           N    C3        'Decrement without steps
 THEN  SET         O0.3      'Counter result decrement
```

Fig. 9/21

### 9.2.3 The counter without the counter (module) [29)]

We admit that this may sound strange. But it is really quite simple.
The count commands DEC – Decrement and INC – Increment can be essentially applied to nearly any word. You can select an output word or a flag word.
Only the comfort of the counter status is no longer available.

However, the counter status sometimes tends to be cumbersome, for example, when multiple counter values should be evaluated or when it is just a matter of an adding a piece counter which should constantly provide information on the parts produced so far.

Let's assume the following task:
A piece counter should show in a traffic light how far the production is.
The count signal comes at the input.
The reset input resets the counter to 0.
RED is shown at the output if the quantity is under 10.
Yellow is shown at the output if the quantity is between 10 and 20.
Green is shown at the output if more than 20 parts have already been produced.

Another note:
A piece counter should usually retain its value even in the event of power loss. We consequently use a variable which is remanent. For the FEC used in the example, this would be a flag from the word range FW0 to FW255.

29) You will find the project described here as Counter_AL.ZIP on your CD.

The program could look like this:

```
"" Reset the piece counter
 IF                I0.1        'Input reset piece counter
 THEN   LOAD       V0
          TO       FW0         'Piece counter

"" Count
 IF                I0.0        'Piece counter signal
        AND    N   F1.0        'Edge detection input signal
 THEN   INC        FW0         'Peice counter
        SET        F1.0        'Edge detection input signal

 IF             N  I0.0        'Signal piece counter
 THEN   RESET      F1.0        'Edge detection input signal

"" Evaluation quantity in red range
 IF                FW0         'Piece counter
          <        V10
 THEN   SET        O0.0        'RED
        RESET      O0.1        'YELLOW
        RESET      O0.2        'GREEN

"" Evaluation quantity in yellow range
 IF             (  FW0         'Piece counter
          <        V20  )
        AND    (  FW0         'Piece counter
          >=       V10  )
 THEN   RESET      O0.0        'RED
        SET        O0.1        'YELLOW
        RESET      O0.2        'GREEN

"" Evaluation quantity in red range
 IF                FW0         'Piece counter
          >=       V20
 THEN   RESET      O0.0        'RED
        RESET      O0.1        'YELLOW
        SET        O0.2        'GREEN
```

Fig. 9/22

## 9.2.4 Combining times and counters [30)]

Examples in which times and counters are combined are also interesting. An example of such a combination is the request to measure the speed of a controller being used, thus a performance measurement.

In a performance measurement we want to check how long an individual cycle lasts. In automation technology there are three values of particular interest for this:

– the cycle time of the controller's central unit, which is completely independent of any sensors or actuators and merely measures the cycle through the programs and modules,

– the cycle time with connected inputs/outputs, thus the reaction time of the system which passes until an input signal switches to an output signal, and finally

– the cycle time of the machine.

In principle all three measurements are performed with nearly identical programs: The counter counts a signal changing once per cycle as 1 upwards, respectively.
Every second (or minute or hour ...) the counter status is read and saved, and the counter starts over again from the beginning. Now we can calculate an individual cycle from the counter status and the time.

30) You will find the program example under Cycle.ZIP on the CD.

## Cycle time measurement

To get started the internal cycle time of the CPU should be measured. For this a flag word is used, which is counted up once in each CPU cycle. Every second the final status is recorded, saved and converted [31].

```
"" The flag word 0 is increased by 1 in each CPU cycle
 IF                NOP
 THEN   INC        FW0        'Is increased by 1 in each CPU cycle

"" A timer is started once every second
 IF            N   T0         'Measure every second
 THEN   SET        T0         'Measure every second
        WITH       1s
""The counter value is saved every second at the same time
        LOAD       FW0        'Is increased by 1 in each CPU cycle
          TO       R0         'Clipboard cycle counter
""And the cycle counter restarted
        LOAD       V0
          TO       FW0        'Is increased by 1 in each CPU cycle

""Evaluation of the counter status
IF                 R0         'Clipboard cycle counter
          <>       V0
THEN    LOAD       V1000
          /        R0         'Clipboard cycle counter
          TO       R1         'Duration of a cycle in ms
```

Fig. 9/23

[31] For your tests please note: Most of the controls have a minimum cycle time. They will not get any shorter, regardless how small your project is. To calculate how much time the controller requires for each instruction, two measurements must be performed. One with X lines of instruction and others with X+Y lines of instruction. The time per instruction can be calculated from the different times.

## Reaction time measurement

For the reaction time measurement, that is, the measurement of the time between when a signal is created at an input until an output is switched over, we will need a little hardware: An input and an output are wired to the controller. Every time the input is switched over in the program, a modified output signal results.

```
"" Switch output
 IF            N    I1.0     'Input for reaction time measurement
 THEN   SET         O0.7     'Output for reaction time measurement
 oTHRW  RESET       O0.7     'Output for reaction time measurement

"" Count
 IF                 I1.0     'Input for reaction time measurement
       AND      N   F2.0     'Edge flag for reaction time measurement
 THEN   INC         FW3      'Reaction time counter
       SET          F2.0     'Edge flag for reaction time measurement

 IF            N    I1.0     'Input for reaction time measurement
 THEN   RESET       F2.0     'Edge flag for reaction time measurement

"" Evaluate
 IF            N    T1       'Time for reaction time measurement
 THEN    SET        T1       'Time for reaction time measurement
       WITH         1s
       LOAD         FW3      'Reaction time counter
         TO         R2       'Clipboard cycle counter
       LOAD         V0
         TO         FW3      'Reaction time counter

"" Convert
 IF                 R2       'Clipboard for reaction time
         <>         V1000
         /          R2       'Clipboard for reaction time
         TO         R3       'Reaction time in ms
```

Fig. 9/24

## 9.3 Limitations of using times and counters

Times and counters are digital elements and use words that are modified. Here FST software also uses words in the sense of PLC, that is, units 16 bits wide. A 16-bit value can – expressed as a decimal – take on values between 0 and $2^{16} - 1 = 65535$.

– As a matter of principle a counter can also count to this value and subsequently begins again from 0. An overrun must be recognised in the program itself, as an overrun bit is not present.

– A time can therefore include up to 65535 clock pulses. As each FST timer works with a clock pulse of 10 ms = 0.01 s, the longest time with a time module can consequently be 65535 * 0 s = 655.35 s = 10 min 55.35 s.

– A time module counts the internal time counter in reverse every 10 ms in increments of 1. This clock pulse is completely independent of all applications and processes. Should a fault arise in the application which should be delayed, an indeterminate period of time between 10 and 0 ms will elapse from this fault up to the first counting pulse. The result of this is that every FST time has an error of up to –10 ms. That is why a time should never be programmed in the program which is smaller than 2 x 10 ms (and this time will then have an maximum error of –10 ms, that is, maximum –50 %).

## 9.4 Practical application of times and counters

### 9.4.1 The garage door with times/counters [32)]

The garage door was programmed in Chapter 3. The Garage project used there should be expanded by two functions:

– The movements of the garage door are counted. After 2000 cycles a display, produced with the aid of an output, informs you that the mechanical parts must be serviced. An additional acknowledgement input resets the counter.

– The limit switch signal 'Door is closed' should be delayed by one second to compensate for inaccuracies which occur when the limit switch is installed.

The program to count the operating cycles is best stored in an independent program. This can look the following way:

```
"" The cycle counter is reset with the acknowledgement input
 IF                I0.7     'Acknowledge cycle counter
 THEN   LOAD       V2000
          TO       CP0      'Set value cycle counter
         SET       C0       'Cycle counter

"" The limit switch signal >Door is open<
"" sends the signal to count the door cycles
 IF                I0.0     'Limit switch garage door is open
        AND    N   F0.0     'Edge detection cycle counter
 THEN   INC       C0       'Cycle counter
        SET       F0.0     'Edge flag cycle counter

 IF           N   I0.0     'Limit switch garage door is open
 THEN   RESET      F0.0     'Edge detection cycle counter

"" After 2000 cycles, a note is displayed to have the machine serviced
 IF           N   C0       'Cycle counter
 THEN   SET        O0.7     'Servicing is necessary
 OTHRW  RESET      O0.7     'Servicing is necessary
```

Fig. 9/25

[32)] You will find this example as Gar_en_T.ZIP on your CD.

The delay of the limit switch is a direct part of the door controller's program. The following is therefore supplemented:

```
""Project: Garage
""Author: Bernhard Plagemann"" Open garage door
 IF              Open_in       'Push button inside Open
       OR    (   Open_out      'Push button outside Open
       AND       Key       )   'Key switch outside
       AND   N   Relay_clo     'Close garage door
       AND   N   Close_In      'Push button inside Close
       AND   N   Close_out     'Push button outside Close
 THEN  SET       Relay_op      'Open garage door

""Stop garage door          'Limit switch garage door is open
       OR        Close_In     'Push button inside Close
       OR        Close_out    'Push button outside Close
 THEN  RESET     Relay_op     'Open garage door

""Close garage door
 IF          (   Close_In     'Push button inside Close
       OR        Close_out )  'Push button outside Close
       AND   N   F0.2         'Door is closed
       AND   N   Relay_op     'Open garage door
       AND   N   Open_In      'Push button inside Open
       AND   N   Close_out    'Push button outside Close
 THEN  SET       Relay_clo    'Close garage door
 OTHRW RESET     Relay_clo    'Close garage door

"" Signal >Door closed< delay
 IF              closed       'Limit switch garage is closed
       AND   N   F0.1         'Edge detection timer ON delay
       AND   N   T0           'Delay door is closed
 THEN  SET       T0           'Delay door is closed
       WITH      1s
       SET       F0.1         'Edge detection timer OFF delay

 IF          N   closed       'Limit switch garage is closed
 THEN  RESET     F0.1         'Edge detection timer ON delay
       RESET     T0           'Delay door is closed

IF           N   T0           'Delay door is closed
       AND       closed       'Limit switch garage is closed
 THEN  SET       F0.2         'Door is closed
 OTHRW RESET     F0.2         'Door is closed
```

Fig. 9/26

# Times and counters with FST in ladder diagram

## Chapter 10

# Contents

There will very unlikely be any automation projects in which times are not used and only a few in which counters are not used. For that reason we should deal with times and counters in detail.

Times and counters belong together, as a time module is a counter that counts clock pulses. However, the time module is somewhat easier to manage than a counter because some counter functions are hidden. Let's start with the times.

## 10.1 The time module [33)]

A time module can be represented as seen below:

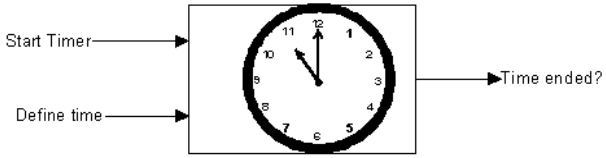



Fig. 10/1: Representation of a time module

A time module must have at least three connections:

– an input for the start

– an input to set the time

– an output that reports whether or not the module is active.

In addition, it is possible to display the elapsed time.

33) You will find all the time examples on the CD entitled Time_LD.ZIP

These inputs and outputs receive a name in FST:

- The time module (timer) itself is called T. As there are 256 timers, you thus have the timers T0 ... T255
- The time is started by calling up this module in a coil.
- The set time, the nominal value, is in the timer preselect, TP0 ... TP255.
- The time module itself becomes '1' after being called up and '0' after the time has expired
- The current time value, the actual value, is in the timer words, TW0 ... TW255.

## 10.1.1 Let's begin with a simple example

When the input I0.6 on our control is actuated, the output O0.6 should be active for a period of 5 seconds:

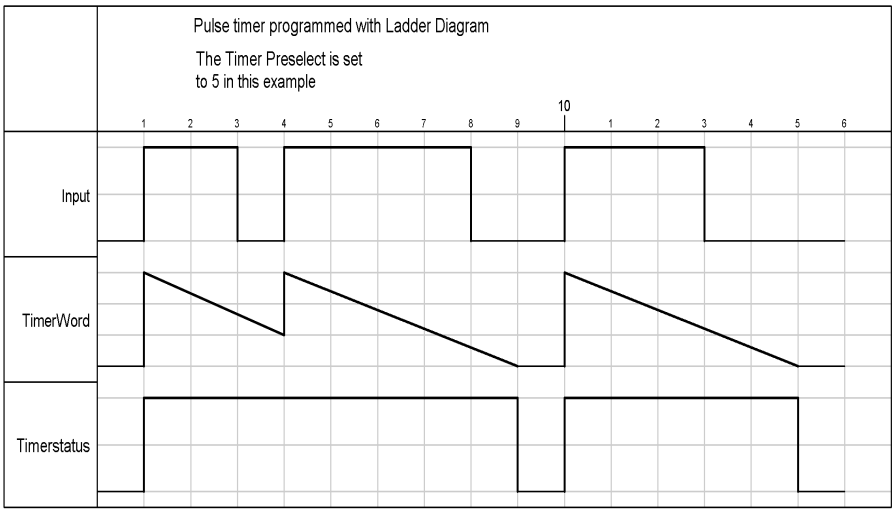



Fig. 10/2

The program can read as follows:

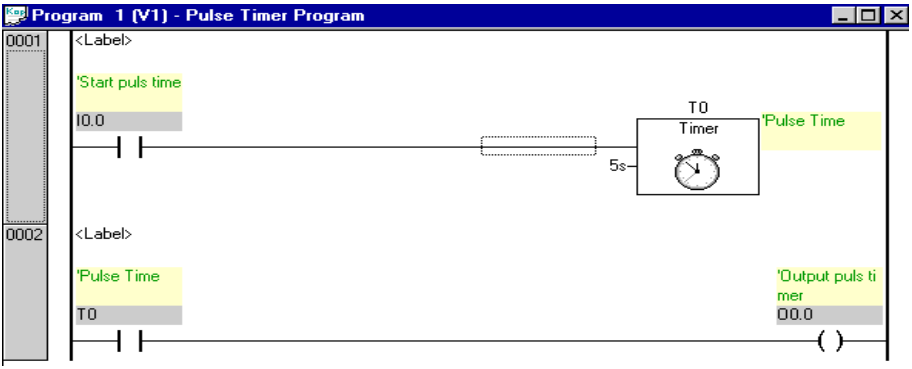



Fig. 10/3

Let's take a closer look at the program:

– The time is started in the first network. Assigning the name 'T0' causes the timer to be used as a pulse timer in the ladder diagram.

– The time is given in seconds at the time input. Here permissible values are 0.01 s ... 655.35 s.

In the online display we see the status, the current value (actual value) and the preselect value (set value):

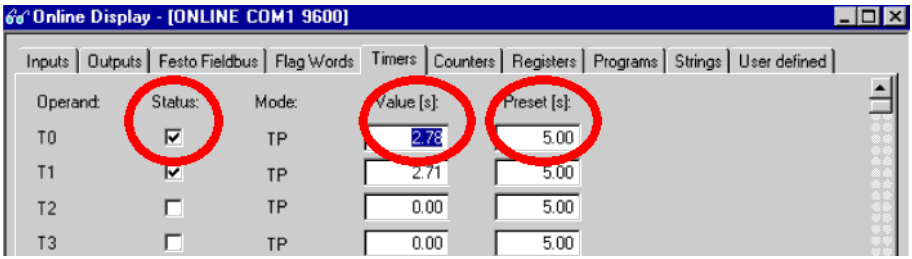



Fig. 10/4

## 10.1.2 The time-on delay

The timer ON or switch ON delay is characterised by the output becoming 1 when the time is over AND the input is still 1. Here is a representation of this:

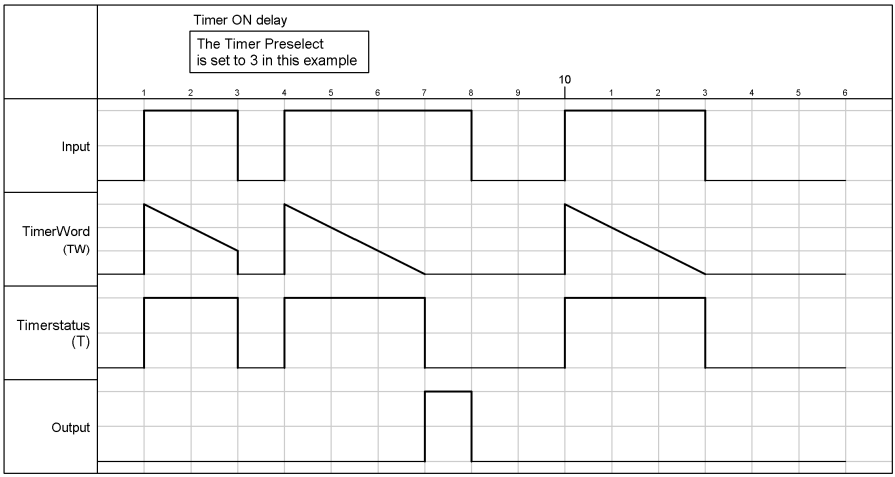



Fig. 10/5

The timer is called 'TON' in the ladder diagram.

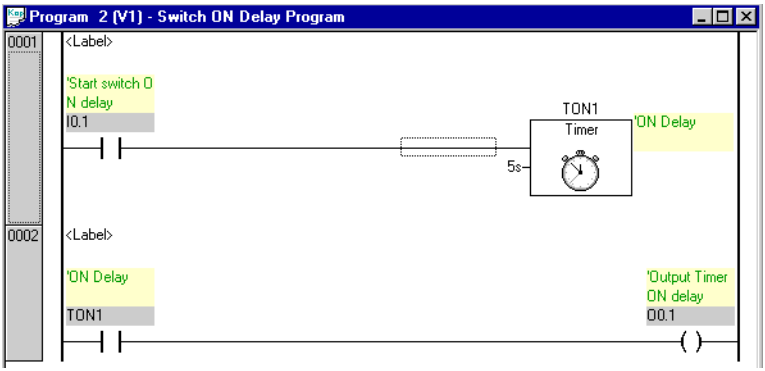



Fig. 10/6

## 10.1.3 The switch OFF delay

The switch OFF or timer OFF delay immediately turns on an output with the input and allows it to be turned on for the preselected time after the input has become '0'.

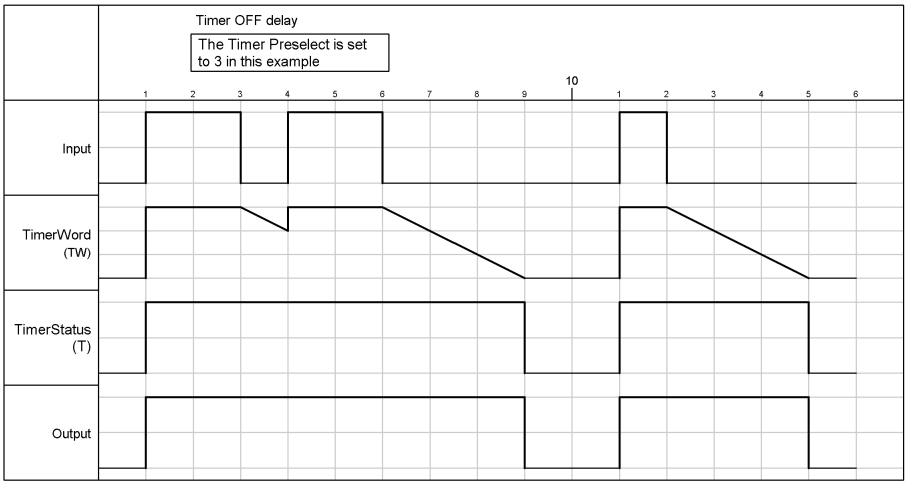



Fig. 10/7

The timer is called TOFF in the ladder diagram – and thus has the characteristics of the switch OFF delay.

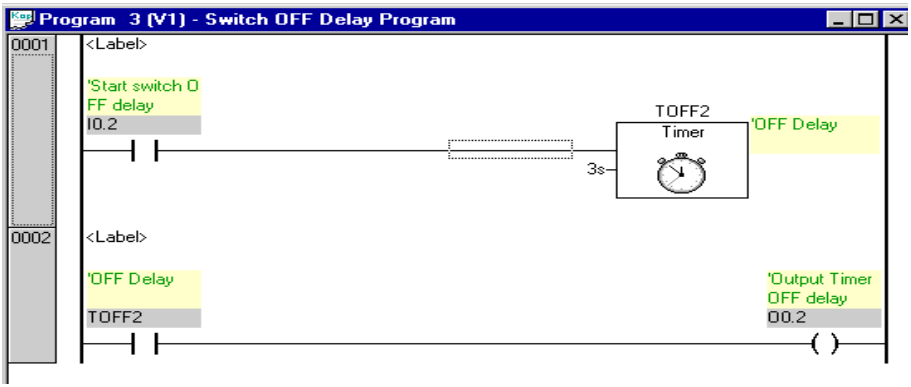



Fig. 10/8

## 10.1.4 The time module in detail

The ready-made time modules do not always suffice. That is why the ladder diagram also allows each individual component of a time module to be addressed individually.
In this way values can be loaded into the timer preselect just as they are into the timer word. This enables a particular time to be modified over the course of a process, for example.
The three components of the time module are:

– Timer status T

– Timer preselection TP

– Timer word TW

The timer status is a bit operand; it can be used in a coil.
Timer preselect and timer word are 16-bit operands; values can also be loaded ('to'), read out, compared, counted up (increment) or counted down (decrement).

The timers always work with a time constant of 10 ms (0.01 s). If we enter the value 50 in a timer preselection, a time of 50 x 10 ms = 500 ms = 0.5 s results.
The following example illustrates direct access to the timer components.

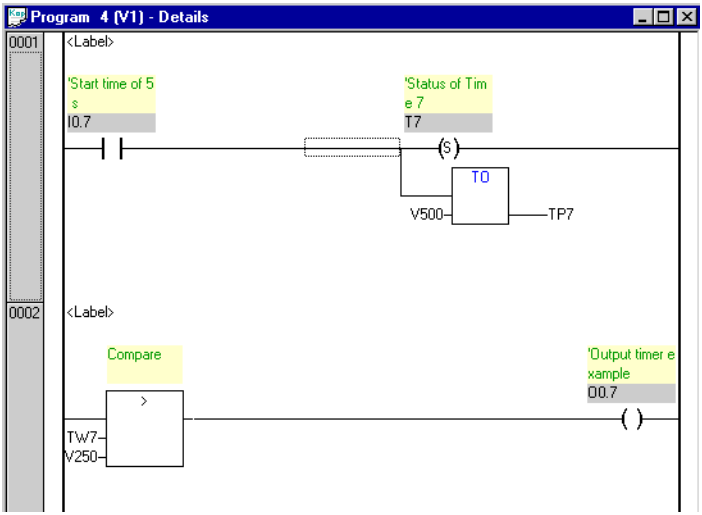

Program 4 (V1) - Details
0001
<Label>
'Start time of 5 s
I0.7
'Status of Time 7
T7
(S)
T0
V500
TP7
0002
<Label>
Compare
>
TW7
V250
'Output timer example
Q0.7
( )

Fig. 10/9

## 10.2 The counter module [34)]

The counter can be viewed in the same way that the time module can be represented. For a comparison with the time module, however, we require additional connections.

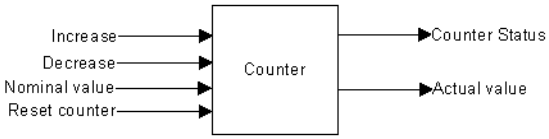



Fig. 10/10

A few practical examples should show the possibilities in handling counters.

### 10.2.1 Count up – increment

Let's assume the following task: The signal at an input should be counted. An output should be turned on after 5 signals. A reset input restores the counter and turns off the output at any time.

[34)] You will find examples of the counter module in the ladder diagram on the CD under Count_LD.ZIP

The program could look like this:

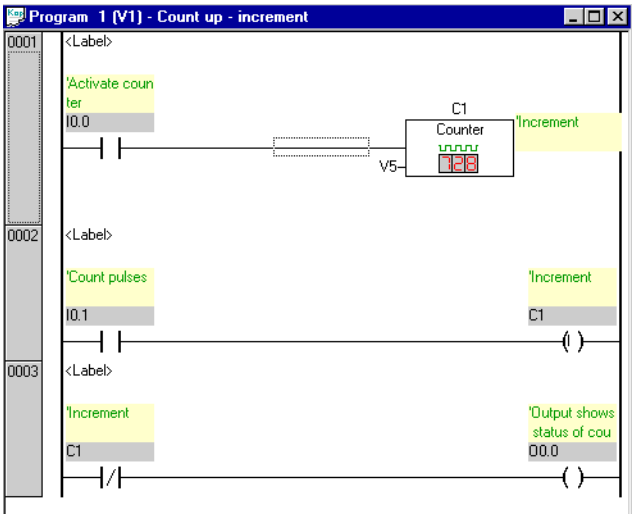



Fig. 10/11

**Please note**
The status of the counter (here C1) is '0' for a new, 'empty' control, i.e. the counter is inactive.
In the program example above this results in the output O0.0 being switched on. In this example, there is no differentiation if the counter becomes '0', because it is only counted up to the counter preselect or because the counter has not yet been activated.

## 10.2.2 The universal counter

Many programmers use the counter modules in a more universal manner. FST can be used to count basically each multi-bit operand with the commands I – increment, count forwards – or D – decrement, count backwards. In the same way, each multi-bit operand can be used in a calcuation (e.g. add +1) and compared (if the value is =25 ...).

The example above could be revised with this condition:

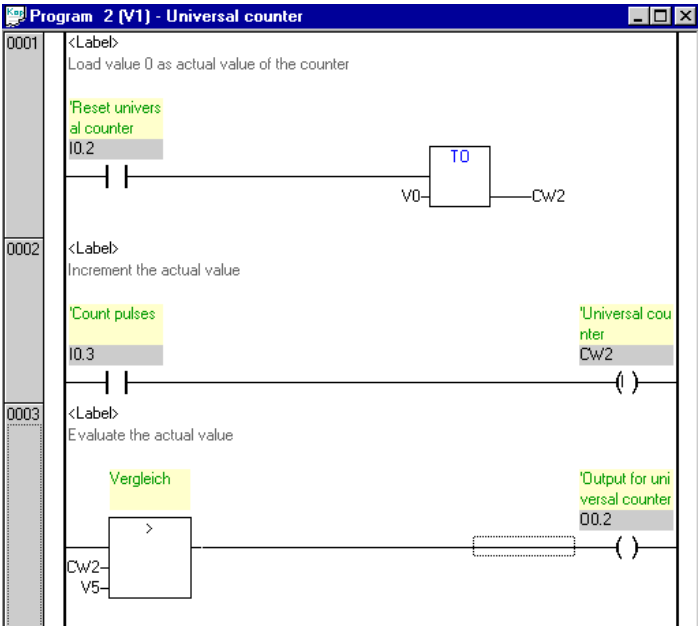



Fig. 10/12

## 10.2.3 Combining times and counters

Examples in which times and counters are combined are also interesting. A well-known example of such a combination is the request to measure the speed of a controller being used, thus a performance measurement [35)].

In a performance measurement we want to check how long an individual cycle lasts. In automation technology there are three values of particular interest for this:

– the cycle time of the controller's central unit, which is completely independent of any sensors or actuators and measures only in the programs and modules,

– the cycle time with connected inputs/outputs, that is, the reaction time of the system that passes until an output signal is switched following an input signal, and finally

– the cycle time of the machine.

In principle all three measurements are performed with nearly identical programs: The counter counts a signal changing once per cycle as 1 upwards, respectively.
Every second (or minute or hour ...) the counter status is read and saved, and the counter starts over again from the beginning. Now we can calculate an individual cycle from the counter status and the time.

### Cycle time measurement

To get started the internal cycle time of the CPU should be measured. For this a flag word is used, which is counted up once in each CPU cycle.

[38)] You will find this example as Cycle_LD.ZIP on your CD.

Every second the final status is recorded, saved and converted [38)].

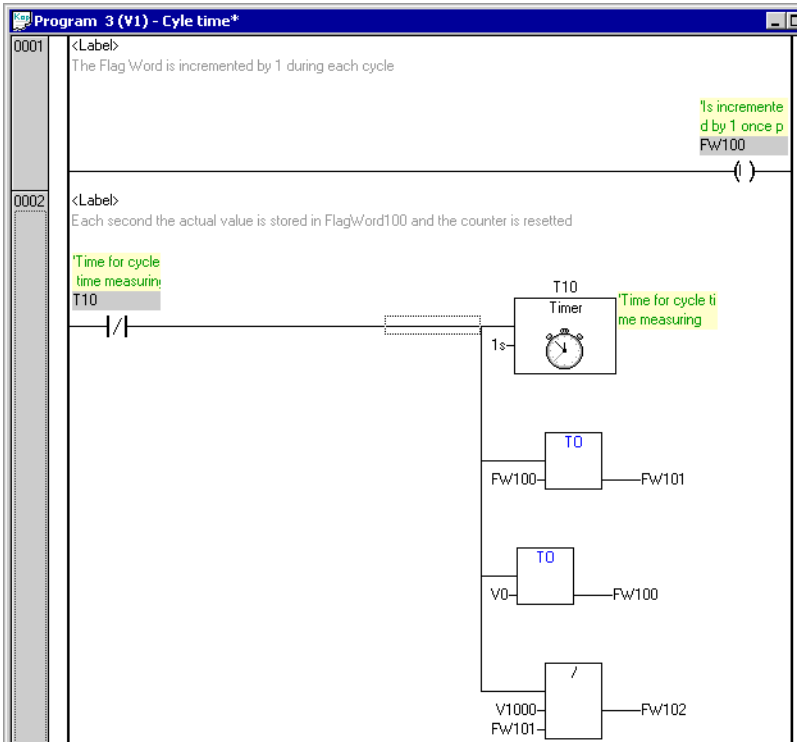



Fig. 10/13

[38)] For your tests please note: Most of the controls have a minimum cycle time. They will not get any shorter, regardless how small your project is. To calculate how much time the controller requires for each instruction, two measurements must be performed. One with X lines of instruction and others with X+Y lines of instruction. The time per instruction can be calculated from the different times.

## 10.3 Limitations of using times and counters

Times and counters are digital elements and use words that are modified. Here FST software also uses words in the sense of PLC, that is, units 16 bits wide. A 16-bit value can – expressed as a decimal – take on values between 0 and $2^{16} - 1 = 65535$.

– As a matter of principle a counter can also count to this value and subsequently begins again from 0. An overrun must be recognised in the program itself, as an overrun bit is not present.

– A time can therefore include up to 65535 clock pulses. As each FST timer works with a clock pulse of 10 ms = 0.01 s, the longest time with a time module can consequently be 65535 * 0.01 s = 655.35 s = 10 min 55.35 s.

– A time module counts the internal time counter in reverse every 10 ms in increments of 1. This clock pulse is completely independent of all applications and processes. Should a fault arise in the application which should be delayed, an indeterminate period of time between 10 and 0 ms will elapse from this fault up to the first counting pulse. The result of this is that every FST time has an error of up to -10 ms. That is why a time should never be programmed in the program which is smaller than 2 x 10 ms (and this time will then have an maximum error of -10 ms, that is, maximum -50 %).

# Sub-programs with FST

## Chapter 11

# Contents

So far you have only become acquainted with parallel programs and not yet with sub-programs. Parallel programs and sub-programs differ in that

– parallel programs, once started, are processed automatically irrespective of whether or not the program which started this parallel program continues to be active.

– sub-programs are processed exactly once and then jump back to the selecting main program. Should a sub-program be processed multiple times, it must be specifically activated each time. A sub-program is also deleted, just like the main program calling it up, if this main program is deleted.

FST allows use of 200 sub-programs, divided into 100 program modules (CMP – Call Module Program) and 100 function modules (CFM – Call Function Module) per project.

Program modules and function programs differ in that

– program modules cause a change of task when called up before they are executed. This means that program modules can also contain steps (in statement list).

– function modules do not cause a change of task when called up. Function modules are **not** allowed to contain steps.

Every sub-program can use input and output parameters. As a matter of principle, up to 7 transfer parameters (to the module) and 7 return parameters (from the module to the program calling up) are available.

There are no local flags within the module. Sub-programs also use the global flags.

FST provides a multitude of modules partially specific to the processor and driver.
In addition, you can create your own modules. This chapter will show you how you can use existing modules.

## 11.1 Importing and naming modules

Not every project uses all the modules possible. You will find an overview of the available modules in the FST documentation and help. If all these modules were present in every project, you would soon lose track. That is why modules are imported into the project before being used.

**Please note**
If the CPU is changed in an existing project, all the modules have to be imported again.

Each module receives its own module number while being imported. You can choose any number between 0 and 99. We recommend always using the same numbers in your own projects.

You can import modules

– from the menu Program, entry Import

– by right-clicking the sub-program CMPs or CFMs, then call Import.

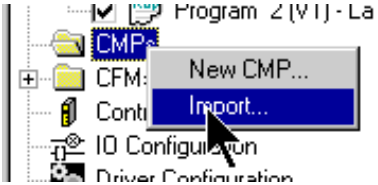



Fig. 11/1

You are then displayed all the modules available for the set CPU.

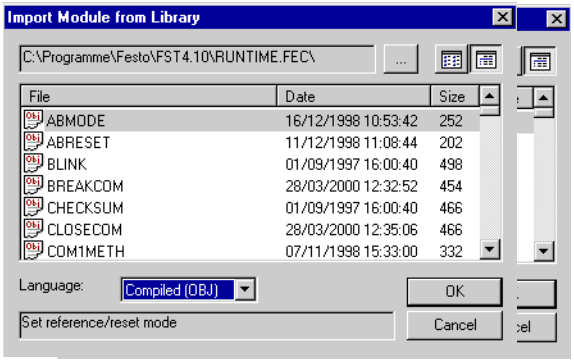



Fig. 11/2

Now select a module and you can give it a type and a number:

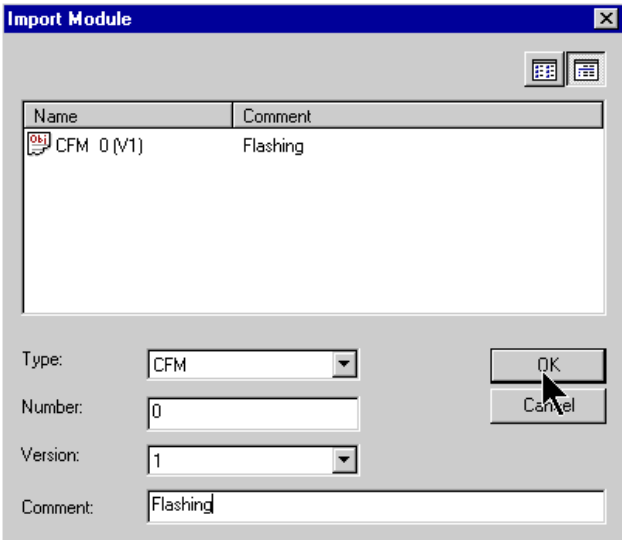



Fig. 11/3

The flash module is imported as CFM 0 in this example. It is available in the sub-directory CFMs.


Fig. 11/4

## 11.2 Transfer and return parameters

Of course, there may be modules that just have something to be taken care – all done. However, a concrete assignment usually has to be transferred to a module and a result is demanded.
There are transfer and return parameters available for this, with up to 7 parameters in each direction. These parameters always have the names FU32 ... FU38.

**Caution**
Transfer and return parameters must always be transferred and called up in direct connection with the module involved. Other modules must not be called up as long as the parameters are neither transferred nor collected.

### 11.2.1 Example of return parameters in statement list

The flash module is a module to which parameters are not transferred but which returns parameters. The flash module sends four flashing bits in the return parameter.
In the following example [37] the flash module was imported first as CFM 0.

```
IF                  NOP
 THEN   CFM 0                   'Flashing
        LOAD        FU32        'First module parameter
          TO        FW0         'Flash bits from flash module
```

Fig. 11/5

[37] You will find this example as Flash.ZIP on your CD. It contains a program in statement list and one in ladder diagram.

The 4 bits with the lowest values flash in flag word 0 at the frequencies 2, 1, 0.5 and 0.25 Hz.

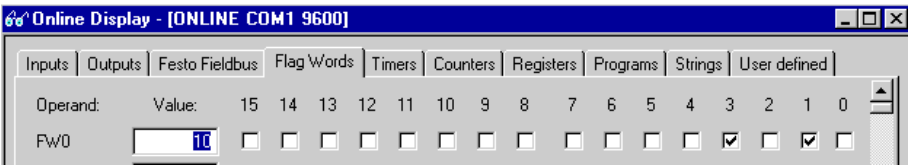



Fig. 11/6

Collection of the return parameters, in the above example

```
LOAD          FU32          'First module parameter
  TO          FW0           'Flash bits from flash module
```

Fig. 11/7

must always take place immediately after the sub-program has been called up. No other modules may be called up in between. The return value must be saved directly in another variable.

## 11.2.2 Example of transfer parameters in statement list

Use the function module F4 to start programs cyclically, every 3 seconds, for example. The program number and the time merely need to be transferred to the module. There is no return parameter.

In the following example [38] module F4 was imported as CFM 1.

```
IF              NOP
 THEN   CFM 1               " Start cyclical program processing
          WITH    V1        " Program number, 0 to 63
          WITH    V3000     " Time in ms; 0:deactivate
```

Fig. 11/8

You see that the transfer parameters are attached to the module call with the aid of the keyword WITH. In this example fixed numbers and constants are used to transfer the transfer parameters.
However, variables could also be used. For example, as follows:

```
IF              NOP
 THEN   CFM 1               " Start cyclical program processing
          WITH    R1        " Program number, 0 to 63
          WITH    R2        " Time in ms; 0:deactivate
```

Fig. 11/9

The transfer values to be transferred are located in both registers R1 and R2 in this example.
The online help of the FST software lists which parameters are to be transferred and in what order. You can receive assistance in entering the parameters. To do this, insert a module by choosing the entry Module Call in the Insert menu.

[38] You will find this example as P_cyc.ZIP on your CD.

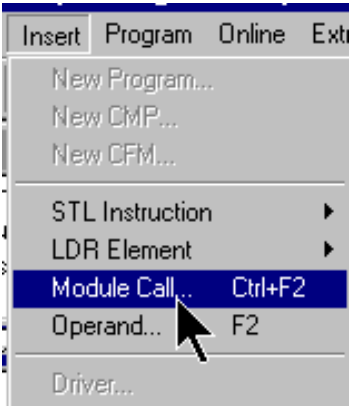



Fig. 11/10

You will be asked which module should be called up

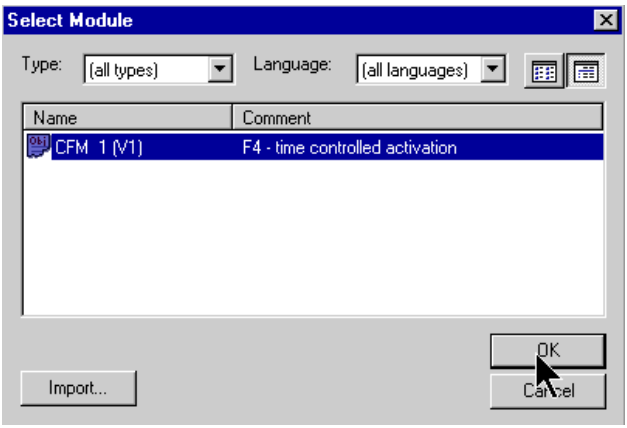



Fig. 11/11

... and then which parameters should be used:

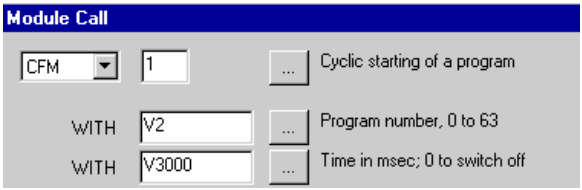



Fig. 11/12

**Please note**
Constants (Vxxx) or multi-bit operands can be used as parameters. The identifier V must precede the constant. In the above example the constant 2 would be transferred for program number 2 and the constant 3000 for 3000 ms.

## 11.2.3 Example of transfer and return parameters in statement list

The module LADD addes two 32-bit values. Every 32-bit value is located in two flag words, as FST flag words are always 16 bits wide. As a consequence, 4 parameters are transferred (both the values to be added) and two parameters are returned (the result).

Let's assume that the numbers 100 423 and 77 000 should be added. The result would be 177 423.

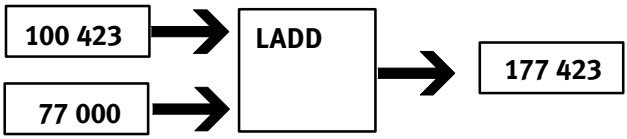



Fig. 11/13

As we must transfer 16-bit words, we have to convert the numbers.

| | **Higher value word** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 423 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| + 77 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| = 177 423 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | **Lower value word** | | | | | | | | | | | | | | | |
| 100 423 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| + 77 000 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| = 177 423 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Tab. 11/1

The 4 individual values – each 16 bits wide – are:

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lower value word from 100 423 | 34 887 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| Higher value word from 100 423 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Tab. 11/2

Add

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lower value word from 77 000 | 11 464 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| Higher value word from 77 000 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Tab. 11/3

Result

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lower value word from 177 423 | 46 351 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Higher value word from 177 423 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Tab. 11/4

In the following example [39] the module LADD was imported as CFM 0.

```
IF                  NOP
 THEN  CFM 0                  " Add 32–bit values
        WITH         FW0       " 1st operand, lower value word
        WITH         FW1       " 1st operand, higher value word
        WITH         FW2       " 2nd operand, lower value word
        WITH         FW3       " 2nd operand, higher value word
       LOAD          FU32      " Lower value word of result
           TO        FW4       ' Lower value word result
       LOAD          FU33      " Higher value word of result
           TO        FW5       ' Higher value word result
```

Fig. 11/14

[39] You will find this example as LADD.ZIP on your CD.

The result of the addition is now located in the flag words FW4 and FW5, see screenshot.

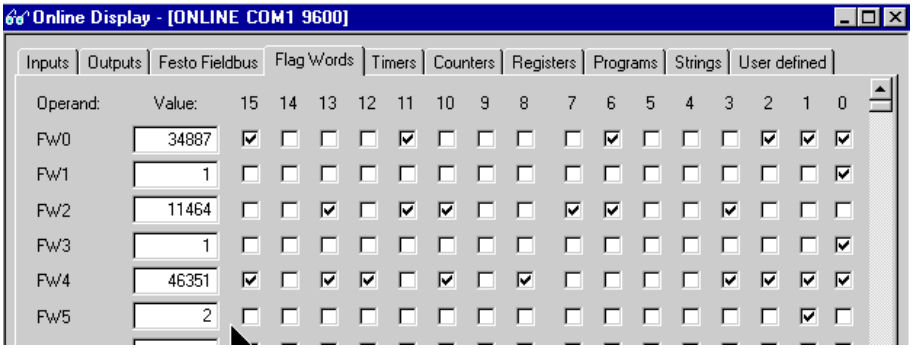



Fig. 11/15

## 11.2.4 Example of return parameters in ladder diagram

The flash module is a module to which parameters are not transferred but which returns parameters. The flash module sends four flashing bits in the return parameter. In the following example [40] the flash module was imported first as CFM0.

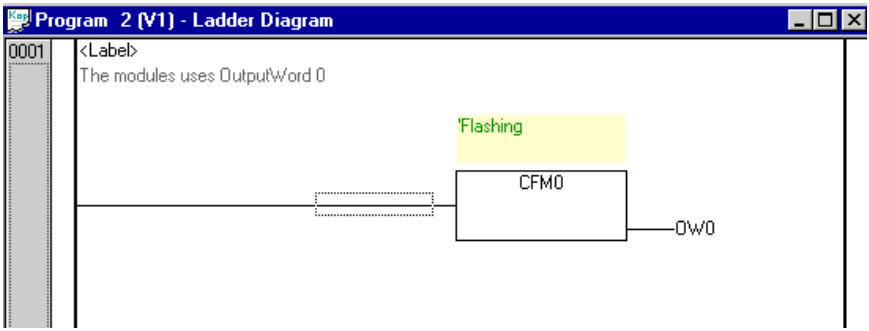



Fig. 11/16

[40] You will find this example as Flash.ZIP on your CD. It contains a program in statement list and one in ladder diagram.

The 4 bits with the lowest values flash in ouput word 0 at the frequencies 2, 1, 0.5 and 0.25 Hz.

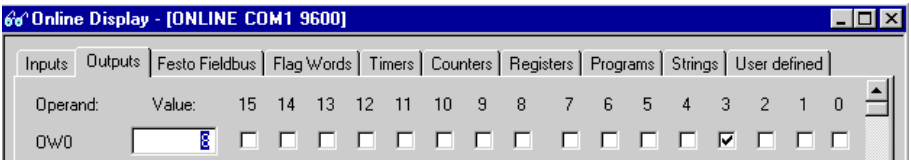



Fig. 11/17

## 11.2.5 Example of return parameters in ladder diagram

Use the function module F4 to start programs cyclically, every 3 seconds, for example. The program number and the time merely need to be transferred to the module. There is no return parameter.
In the following example module F4 was imported as CFM1.

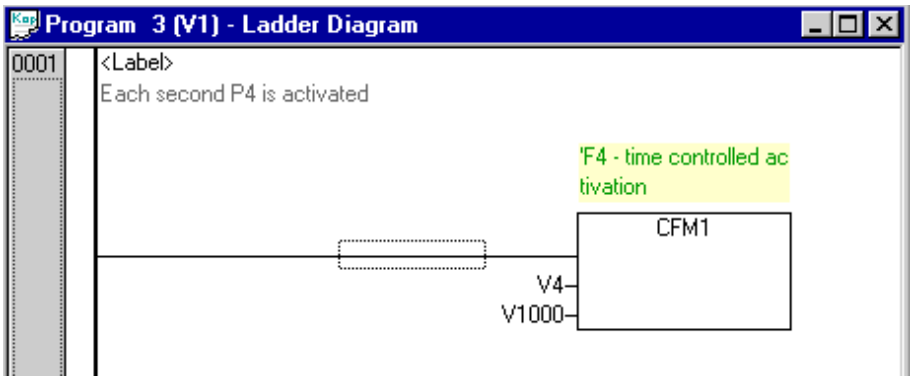



Fig. 11/18

In this example fixed numbers and constants are used to transfer the transfer parameters.
However, variables could also be used. For example, as follows:

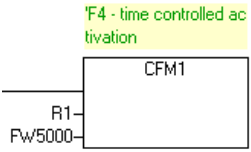



Fig. 11/19

The transfer values to be transferred are located in registers R1 and flag word 5000. The online help of the FST software lists which parameters are to be transferred and in what order.

**Please note**
Constants (Vxxx) or multi-bit operands can be used as parameters. The identifier V must precede the constant. In the above example the constant 4 would be transferred for program number 4 and the constant 1000 for 1000 ms.

## 11.3 Sub-programs with FST: Creating your own modules

You can create your own sub-programs like normal programs. They can even also use transfer and return parameters.

### 11.3.1 My own shift module

A module [41] is supposed to shift the bits in a word by a specified number of places. The module requires three transfer parameters and a return parameter. The transfer parameters are:

- the word being shifted
- the number of places that it should be shifted
- the direction of the shift (1 = right, 0 = left).

The return parameter is the finished word.

The module call is quite simple in program performing the call:

```
"" This program calls up a module representing a shift register
 IF                  I0.0          'Input 0.0
       AND      N    F0.0          'Edge detection
 THEN  SET           F0.0          'Edge detection
       CMP 0                       'My shift register
        WITH         FW1           'The flag word to be shifted
        WITH         V3            "Shift three times
        WITH         V0            "Right=1, Left =0
       LOAD          FU32          'First module parameter
          TO         FW2           'The result of shifting

 IF            N     I0.0          'Input 0.0
 THEN  RESET         F0.0          'Edge detection
```

Fig. 11/20

[41] You will find this example as My_CMP.ZIP on your CD.

The shift register itself is now written in the CMP 0 as follows:

```
""I wrote this program module myself
STEP Start
"" The transfer parameters are written in flag words first
 IF                  NOP
 THEN   LOAD         FU32          'First module parameter
          TO         FW10          'The value to be shifted
        LOAD         FU33          'Second parameter
          TO         FW11          'Shift how often?
        LOAD         FU34          'Third parameter
          TO         FW12          'Right = 1, Left = 0

STEP Shift
"" Then shift as long as demanded
 IF             (    FW11          'Shirft how often?
          >           V0      )
        AND     (    FW12          'Right = 1, Left = 0
         =           V1      )
 THEN   LOAD         FW10          'The value to be shifted
         SHR
          TO         FW10          'The value to be shifted
        DEC          FW11          'Shift how often?

 IF             (    FW11          'Shirft how often?
          >           V0      )
        AND     (    FW12          'Right = 1, Left = 0
         =           V0      )
 THEN   LOAD         FW10          'The value to be shifted
         SHL
          TO         FW10          'The value to be shifted
        DEC          FW11          'Shift how often?

 IF                  FW11          'Shirft how often?
          =          V0
 THEN                NOP

STEP Forward
""Finally, the new value is written in the return parameter
 IF                  NOP
 THEN   LOAD         FW10          'The value to be shifted
          TO         FU32          'First module parameter
```

Fig. 11/21

## 11.4 Difference between CFM/CMP

As represented above, CFM and CMP differ only in that a CMP causes a change of task when called up and a CFM does not. The task change principle is represented once more as a reminder:

- Up to 64 programs can be created with FST. Each one can be activated and thus becomes a task.
- The processor switches among these tasks:
  - in ascending order of the program number – therefore from P0 to P63
  - at the exact point when a complete step has been fully processed or at the exact point when a return jump occurs in the ladder diagram
  - at the exact point when a CMP is called up.

Calling up a program module (CMP) causes a change of task. You can also use steps within a program module. They also each cause a change of task.

Calling up a function module (CFM) does not cause a change of task. **No** steps may be used within a function module, as the function module would never come any further than the first step within the module.

# Recognising errors with FST: The FST error program

## Chapter 12

# Contents

## 12.1 General information on errors in a FST system

Every controller knows internal error messages. You will find a list of the FST error numbers in the appendix. Such an error is reported in the 'error word' and displayed in the online control panel. The following is displayed in doing so:

| Error type | Composition of the CI reply |
|---|---|
| General error | =‹Error number›,‹program number›,‹step number› 1) |
| CPX error (42) | =‹42›,‹CPX error number›,‹CPX module number› |
| I/O error (11, 12) | =‹Error number›,‹255›,‹number of the input or output word› |
| 1) The error number corresponds to the value of the error word; the program number in which the error appeared; should the program not have any steps (e.g. for LDR programs), step 0 is displayed. | |

Tab. 12/1

If the error did not appear while a program was being processed, then Program No. 255 is displayed. If the program does not have any steps – for example, every ladder diagram program – Step No. 0 is displayed.

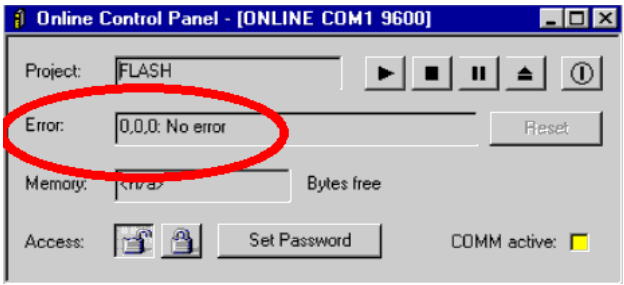



Fig. 12/1

Both the error word (EW) and the status bit E can also be shown in the online display.

Please observe: You will find the error word displayed in the 'User defined' area of the online display. Enter the EW or E as operand there.

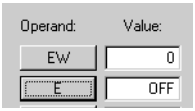



Fig. 12/2

The controller automatically changes the error word as an error has been registered. The following image shows error message 6 in both the control panel and the online display:

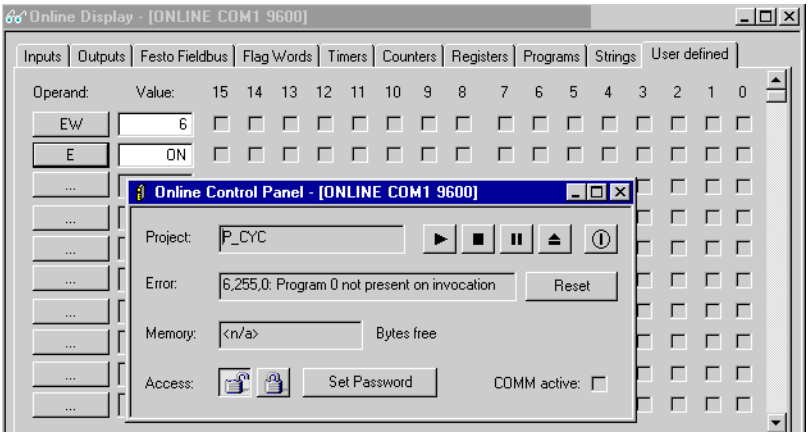



Fig. 12/3

The error word is also available in the user program. With

```
THEN    LOAD    V133
        TO      EW
```

or

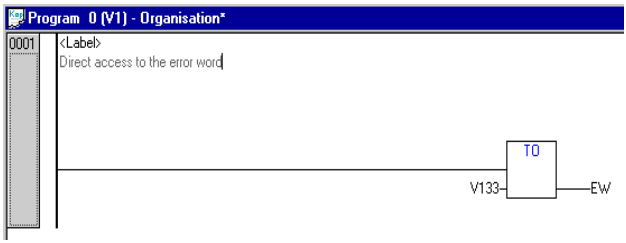



Fig. 12/4

error no. 133 is produced.

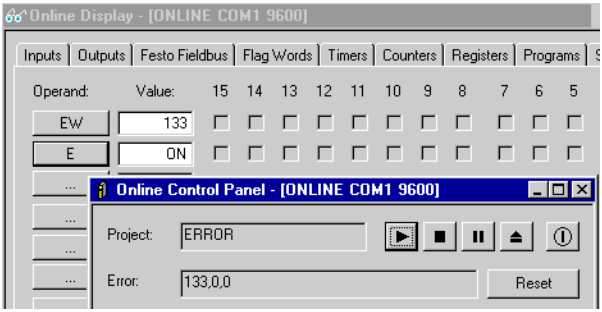



Fig. 12/5

## 12.2 The reaction to an error

The normal reaction without any change in the basic setting is as follows:

As soon as the content of the error differs from 0, that is, as soon as the error status has switched to ON,

– the controller switches to Stop,

– the RUN LED turns red and

– the outputs switch off.

This reaction is called a 'hard error reaction'.

Using Controller Settings, Run Mode the following two features can be modified independent from each other:

– An error output switches on when an error occurs. Which output can be freely set and/or

– the outputs remain switched on.

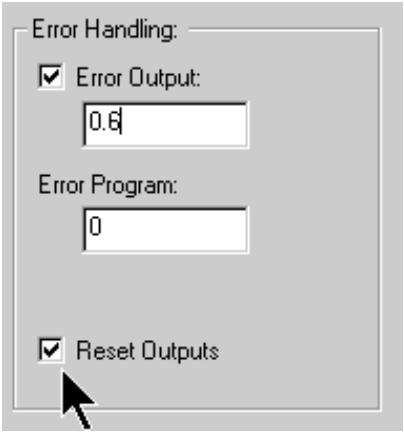



Fig. 12/6

In the process the controller can still be reached from outside. In an operator terminal, for example, the error word can be read out and interpreted for trouble shooting.

Only the very first error that occured is saved. Any subsequent errors are ignored.

When a project is downloaded the error word is automatically deleted.

Should the controller itself react to an error in any form whatsoever, it must be able to process a program. That is what the error program is for.

## 12.3 The error program

The error program can be entered under 'Controller Settings' in the tab Run Mode.

Entering in the error program means:

– Error program 0 means that there is not an error program.

– Each program number from 1 ... 63 means that the program entered is started in the event of an error.

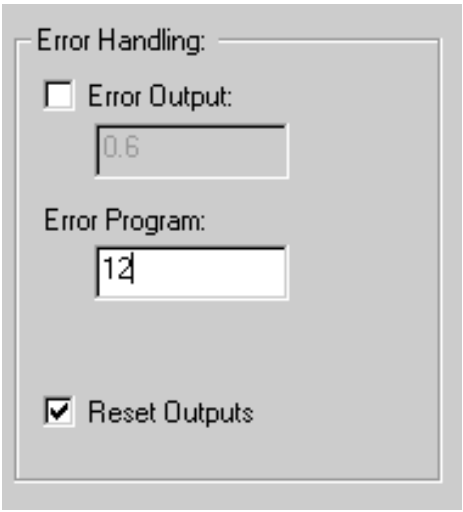



Fig. 12/7

### 12.3.1 The error reaction with the error program

If an error program exists then the following will occur in the event of an error:

– the program in which the error occurred is stopped,

– all other programs are halted (they are not stopped but rather 'doze') and

– the error program is activated.

The error word is evaluated in the error program. Afterwards the following can be done:

– the error can be deleted and the rest of the programs can be activated again or

– all the outputs and programs can be reset (major errors).

To be able to react to the error, you require is module F22.

You can import module F22 from the FST module library into the current project (see also section 11.1).

F22: Setting the error handling
Input parameters

| | | |
|---|---|---|
| FU32 | = 0 | Query error data |
| | = 1 | Query error data and delete errors |
| | = 2 | Continue processing program, Query error data and delete errors |
| | = 3 | Errors that cannot be rectified, Stop program |

Return parameters

| | |
|---|---|
| FU32 | Error number |
| FU33 | Program number |
| FU34 | Step number |
| FU35 | Always 0 (error address) |

## 12.3.2 The garage door with error program [42)]

Let's assume that the limit switches for the garage door program should be monitored. If both limit switches transmit a 1 signal at the same time, there must be an error. This error should be signaled with the output O0.5 until the error is no longer present. The programs should then be 'woken' again, that is, the controller continues to work.

The monitoring program P2 is supplemented for this:

```
"" Monitor limit switches
 IF              open      'Limit switches garage door is open
        AND      closed    'Limit switches garage door is closed
 THEN   LOAD     V100
          TO     EW
```

Fig. 12/8

Error 100 is thus the new error number for a defective limit switch in the garage door drive.

As soon as the error occurs, the error program is activated and the rest of the programs 'sleep'. The error is displayed in the upper right of the online window in online mode:


Fig. 12/9

[42)] You will find this program example as Gar_Err.ZIP on your CD.

The error program differentiates between system errors and limit switch errors:

```
"" System error which is not programmed
"" Hard reaction
 IF           N (    EW
        =           V100    )
 THEN  CFM 22                    " Setup error handling
         WITH       V3           " 0:query; 1:query+del;
                                 " 2:query+del+continue; 3:stop
       RESET        P12

"" Limit switch error
"" Wait for the limit switch to be repaired
 IF             (    EW
        =           V100    )
       AND      (    open        'Limit switches garage door is open
       EXOR          closed  )   'Limit switches garage door is closed
 THEN  CFM 22                    " Setup error handling
         WITH       V2           " 0:query; 1:query+del;
                                 " 2:query+del+continue; 3:stop
       RESET        P12
```

Fig. 12/10

## 12.3.3 Description of an error

Known errors are displayed with a text in online mode. This text is taken from the file erroripc.txt. This file is located in the following directory:

– English version: \FST4

– German version: \FST4\DEU

Here you can supplement the following, for example:

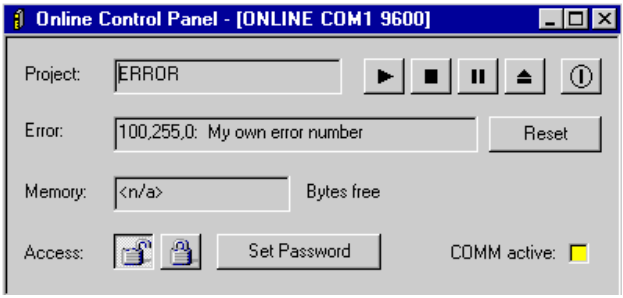



Fig. 12/11

# Operating with FST: Connecting and programming operator terminals

## Chapter 13

# Contents

Hardly any machine can get by these days without operator terminals. The more sophisticated the design of the operator guidance, the easier it is to train operating personnel. The more creativity and time invested in the design of debuggers, the faster a defective proximity switch can be found and exchanged in case of emergency – every minute of downtime saved translates into great savings.


Fig. 13/1: FED operator terminal from Festo
(here type FED 50)

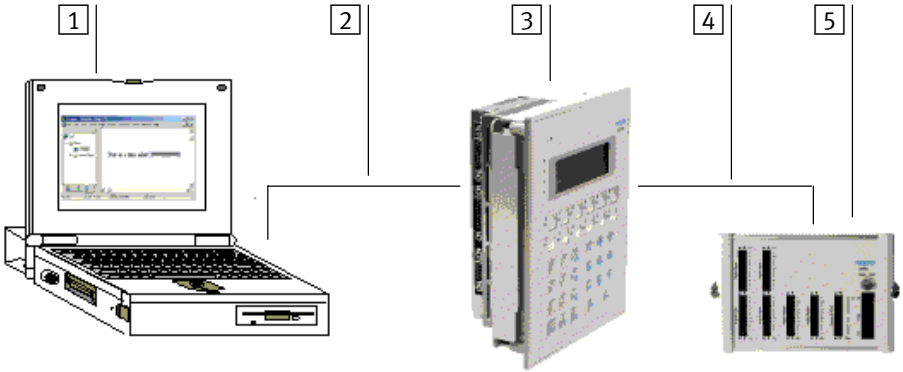
In this chapter the foundation is laid for connecting the Festo FED operator terminal.

## 13.1 Connecting the operator terminal

Operator terminals are usually connected via a serial port.

For CPX-FEC, plug in the FED using the connection cable type FEC-KBG7 or FEC-KBG8 (article no. 539 642 or 539 643). FEDs have a 15-pin Sub-D interface for connecting to the controller (PLC interface). A cable for connecting to the controller is required, depending on the CPU. The KBG3, TN 189 429 cable is required for the COM port of the IPC FEC FC20, FC21, FC22, FC23.
A KBG6, article no. 189 432 is used for the COM port of the HC16 CPU or the CP3X modules, whereby the adapter included with the cable is unnecessary and can be disposed of.

The KBG6, article no. 189 432 is required for all the remaining interfaces, that is, for FC2X CPUs (EXT port), FC3X CPUs, all IPC FEC standard interfaces, HC0X CPUs and HC20 CPUs.




1 PC with FED Designer
2 Serial connection
3 Front end display (FED)
4 Serial connection/Ethernet
5 Controller

Fig. 13/2

### 13.1.1 The EXT port in the FECs (front end controller)

If a display is connected to the EXT port of a FEC, it must be activated first. For this:

- The COMEXT driver must be integrated in the driver configurarion.

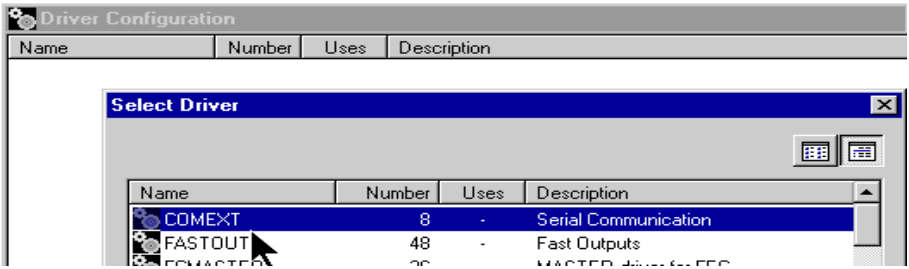



Fig. 13/3

- The EXT port must be opened in the program and the CI access activated. The modules OpenCOM and F31 (activate CI) are required for this.


Fig. 13/4

The EXT port is now opened in an initialisation step [43)] and the CI is activated on this interface.

[43)] You will find this program on your CD in the allocation list as FED_EXT.ZIP, in the ladder diagram as FED_ELD.ZIP.

```
STEP Init
"" Open the EXT serial port
"" ATTENTION: The COMEXT driver must be entered in the driver
"" configuration!
 IF                NOP
 THEN  CFM 0                  " Open serial port with 9600, 8, N
         WITH      V0         " Serial port

IF                 FU32       'First parameter for modules
         =         V0
 THEN              NOP

STEP
"" Activate the CI on this port
 IF                NOP
 THEN  CFM 1                  " Activate CI
         WITH      V0         " Serial port
         WITH      V2         " 0=disable CI (Default); 1=enable CI
```

Fig. 13/5

The display can now get the necessary data via the EXT port.

Please observe:
The module F31 used to activate the command interpreter CI receives two transfer parameters.

– The first one sets the interface used. Use No. 0 to select the EXT port for the FECs.

– The second parameter defines the reaction of the command interpreter. V2 should be entered here for communicating with the FECs to prevent the interface from the FED being accidentally turned off.

## 13.1.2 Communication between FED and the programming PC

Of course, a project, which is created with the FED Designer described later, must be loaded into the FED. The cable FEDZ-PC (article no. 533 534) is required for communication between the PC and the display.

If you would like to have access simultaneously to the controller and the FED and also use a COM port for the controller, then set the software FED Designer to COM2 and the FST to COM1 or vice versa. You will find the setting in the FED Designer in the menu Transfer, Options:

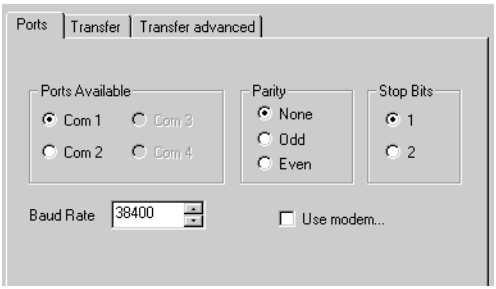



Fig. 13/6

To be able to communicate with the display from the PC, this must be switched on in the configuration mode.
To do this, actuate the Enter key on the display until a system menu appears on the display. Select the entry CFG using the cursor keys and confirm by pressing Enter. Now – in “Configuration mode” – the display can communicate with PC but no longer with the controller.

## 13.2 FED Designer and FST

The FED Designer is used to program the FEDs [44]. The software is an independent Windows application. Using the FED follows simple rules:

Start FED Designer [45]:

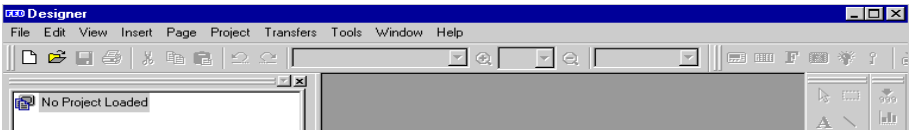



Fig. 13/7

Start a new project using the menu File; New or click on the New icon.

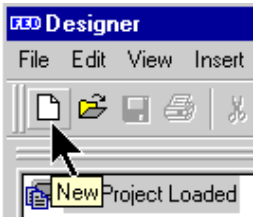



Fig. 13/8

[44] FED – front end display

[45] You will find the FED Designer project on your CD under the name FED_new.dpr

Enter the project name.

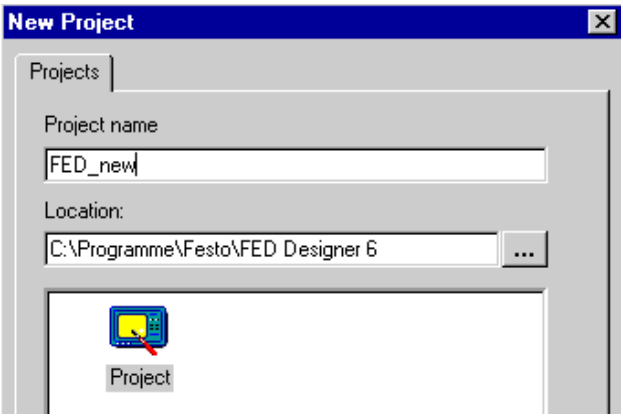



Fig. 13/9

Set the display type or read out the display being used via the menu Project, Panel Setup.

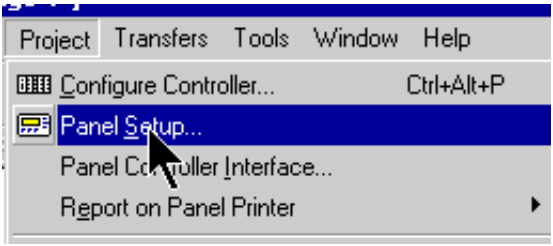



Fig. 13/10

Select a display

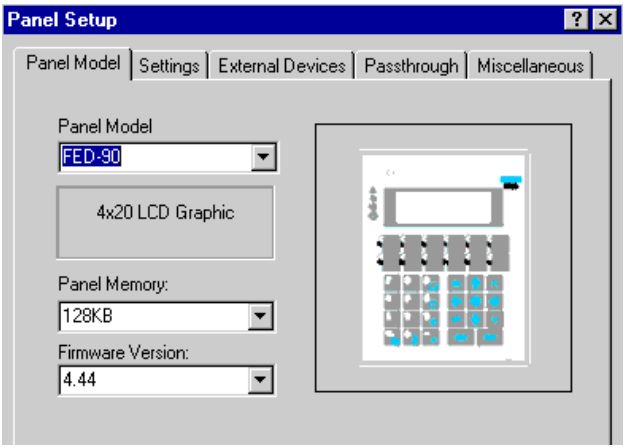



Fig. 13/11

## 13.2.1 Showing a variable on the display

Let's assume that the number of parts already produced is saved in flag word 10 of the controller. This value is shown on the display. An explanatory text is placed ahead:


Fig. 13/12

Now select a numeric value for the PLC

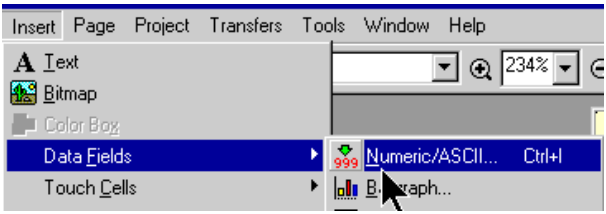



Fig. 13/13

The cursor turns into a cross-hair with which you can pull up the area to display the controller's value.

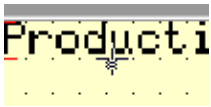



Fig. 13/14

As soon as you release the mouse button, the input window opens for the value to be transferred:

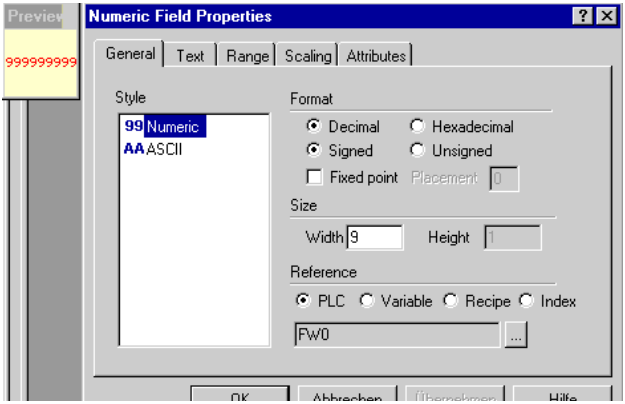



Fig. 13/15

Here you can enter the data to be displayed. This should be a flag word in our example.
What is more, you can also modify the field width. A 16-bit value without any preceding signs requires at least 5 digits (e.g. 65000) for the maximum value.

On the FED Designer display you will now see a substitute symbol for the value which will be retrieved later from the PLC:


Fig. 13/16

## 13.2.2 Modifying a variable from the display

Let's assume that the nominal value for time is saved for a while in flag word 11. This time should be able to be modified from the display. To do this call up the 'Range' page in the top of the window already being used and enter that it is permissible to read and write, as well as – if it is reasonable – the range within which a value is permissible.

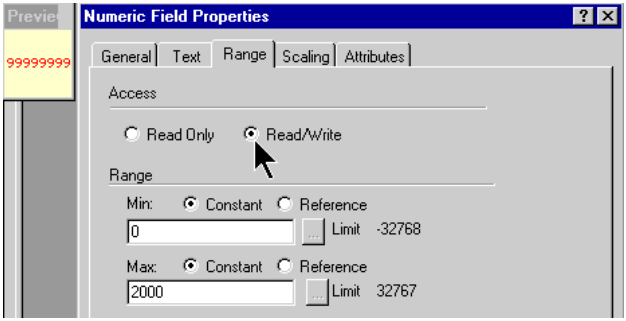



Fig. 13/17

The following could then appear on the display:


Fig. 13/18

When the project has been downloaded into the FED and the FED has been started, you can modify the nominal value by pressing the → key. The FED switches to data entry mode and the value can be modified within the specified limits. To do this, enter the value you want using the numerical keys and press Enter to confirm.

It is possible that some FEDs do not have this key. This function must then be allocated to a function key or a touch cell using the keyboard macro editor.

## 13.3 The FED in the Ethernet network

Chapter 14 describes how the FST programmable controls are integrated in a network. Of course, FEDs are also part of this network.

The prerequisite for this is acquiring the Ethernet interface for the FED including the adapter plug, designation FEDZ-IET, article no. 533 533. Attach the Ethernet module as per the assembly instructions.

### 13.3.1 The FED Designer project for Ethernet communication

For communication via the network, the FED Designer project must be informed that Ethernet should be used instead of the usual PLC interface. To do this, call up the entry “Configure Controllers” in the Project menu.

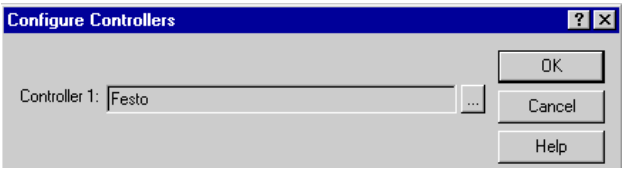



Fig. 13/19

Open the controller by clicking on the menu button:

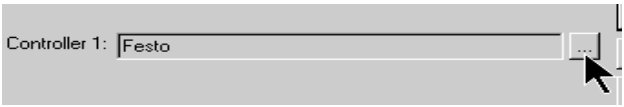



Fig. 13/20

Select Festo Easy IP and open the Controller setting.

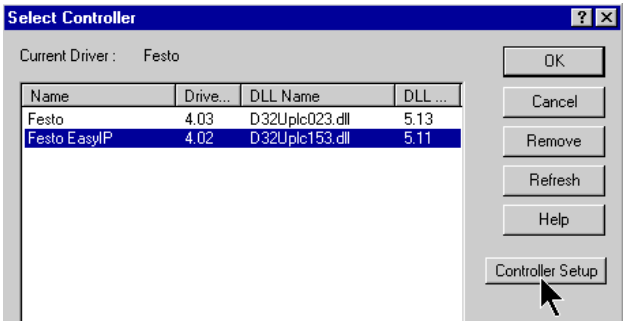



Fig. 13/21

**Please note**
Enter the IP address of the controller (not the FED) here with which the FED should communicate. The controller must receive this IP address by means of the FST project.

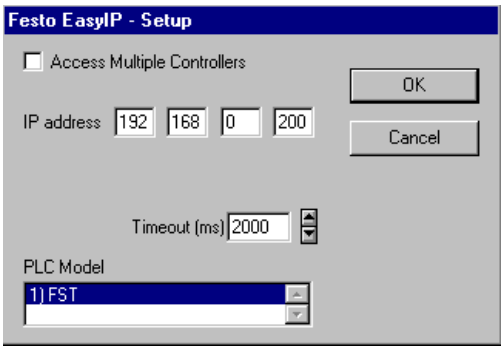



Fig. 13/22

In addition, the FED itself must receive an IP address. To do this, call up the entry “Panel Setup” in the Project menu. Once there, select the rider “External Devices” and enter the panel’s IP address.

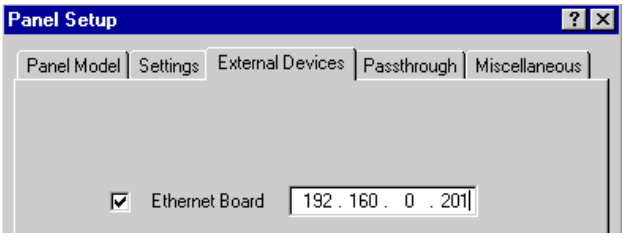



Fig. 13/23

**Please note**
Enter the IP address of the FED (not the controller) here.

As soon as

– the project has been downloaded into the FED,

– the Ethernet connections have been established,

– the controller has received a project with the TCPIP driver and the corresponding IP address,

communication takes place between the FED and the controller via the network.

## 13.3.2 Multiple controllers in the network with a FED

Multiple controllers can communicate with a FED with the aid of the Ethernet network. To do this access to more than one controller “Access Multiple Controllers” must be entered in “Configure Controllers”.

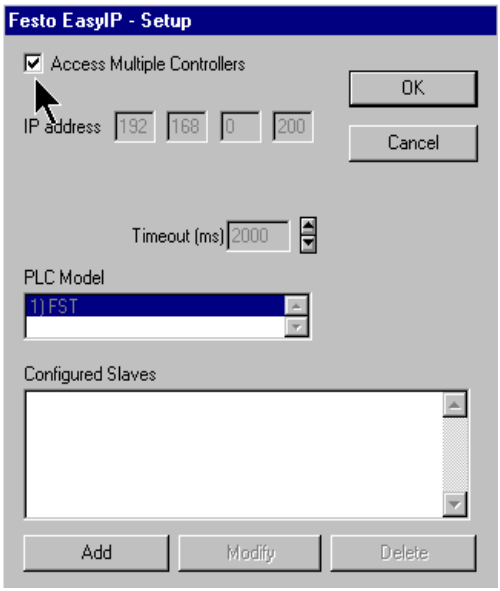



Fig. 13/24

Clicking “Add” allows all the existing controllers in a network to be entered consecutively.

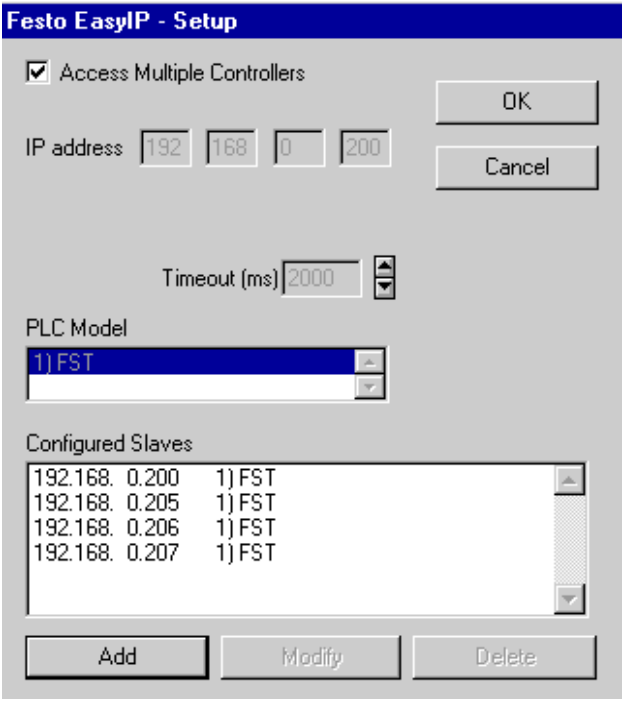

Festo EasyIP - Setup
Access Multiple Controllers
OK
IP address
192
168
0
200
Cancel
Timeout (ms)
2000
PLC Model
1) FST
Configured Slaves
192.168. 0.200 1) FST
192.168. 0.205 1) FST
192.168. 0.206 1) FST
192.168. 0.207 1) FST
Add
Modify
Delete

Fig. 13/25

Should a variable of a controller now be accessed by FED, this controller must be named. With Insert, Data Fields, Numeric, for example, the window already used is opened.

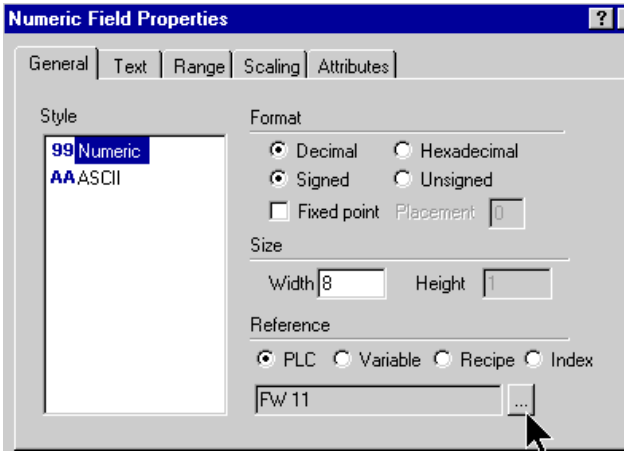



Fig. 13/26

The button directly next to the variable address now opens not only the selection of the variable but also the controller selection.

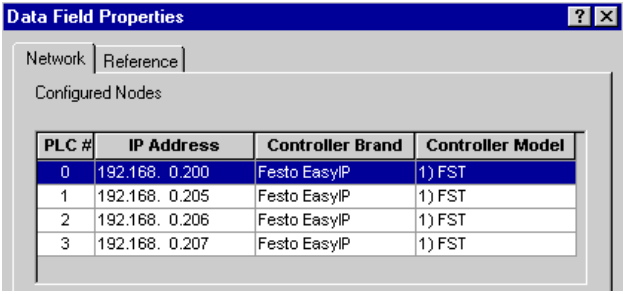



Fig. 13/27

# Networking with FST

## Chapter 14

# Contents

Communication for automation technology has never been as important as it is today. And we're not talking about collecting and distributing input/output data. Fieldbuses have been taking care of that for a long time in a fast, secure manner. What is meant is communication between the controllers and between the Windows office world and and the controllers in production. Ethernet, the network standard at the office, has proved itself to be the ideal medium for communication, even in production. Use of Ethernet is simple and secure with FST.

## 14.1 Prerequisite for using Ethernet – the TCP/IP driver

Every FST CPU can be supplied with an Ethernet interface. If the hardware requirement – the Ethernet interface – is specified, the software requirement must also be met. This is the TCP/IP driver.

The TCP/IP driver is entered in FST as are all drivers in the driver configuration. To do so, open the 'Driver Configution' by double-clicking.

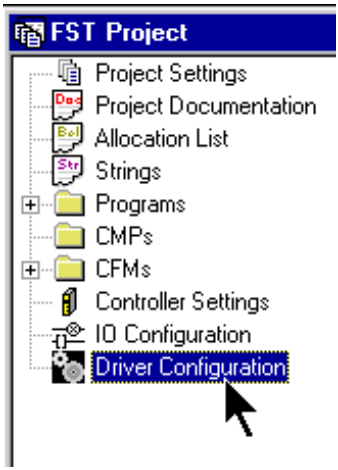



Fig. 14/1

The TCP/IP driver has already been pre-installed at the factory for CPX-FEC.

In the driver configuration window you can call up the drivers available on this CPU by double-clicking on an empty location.

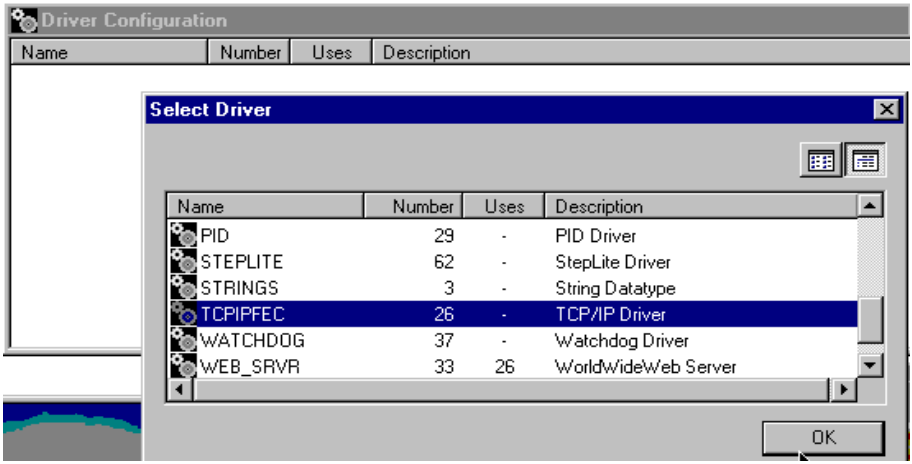



Fig. 14/2

Please note – as with all drivers – that drivers are CPU-specific [46]. If you change the CPU during a project, all the drivers must be inserted anew into the project.

[46] The TCP/IP driver for the FEC Compact is TCPIPFEC, and for the FEC Standard it is TCPIPFC2.

The addresses in particular are important in the following dialog:

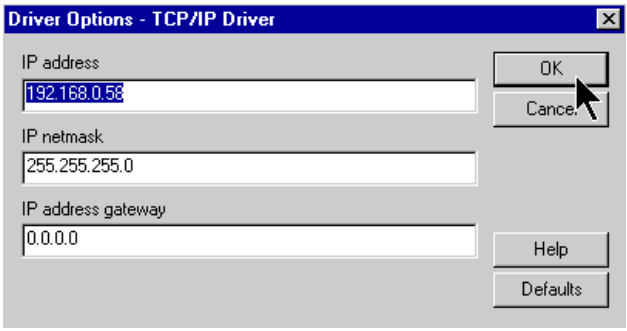



Fig. 14/3

IP address

An IP address ic composed of 4 bytes, that is, 4 numbers between 0 and 255.
The addresses 0.0.0.0 and 255.255.255.255 are not permissible. You can freely select the address in a closed, local network consisting exclusively of your controllers and your programming PC. If your network is connected to your company's network, addressing must be arranged with the IT department.

IP netmask

The mask limits usage of the addresses. Each and every bit of an IP address corresponds to a bit in the network mask. If the bit in the network mask is '1', then the bit in the IP address must be identical between the two partners communicating.

IP address of the gateway

Should your controller network be connected to another network with the aid of a gateway, this gateway's address must be entered here. If you are not using a gateway, the gateway address can remain 0.0.0.0.

### 14.1.1 Excursion into IP address and network masks

In the above example the controller's IP address is stated as:
168.192.0.3

It is written in binary as:
1 0 1 0 1 0 0 0 . 1 1 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . 0 0 0 0 0 0 1 1

The usual network mask for sub-networks in automation is:
255.255.255.0

It is written in binary as:
1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 . 0 0 0 0 0 0 0 0

The address of an arbitrary communication parter without using the gateway must be as follows:
1 0 1 0 1 0 0 0 . 1 1 0 0 0 0 0 0 . 0 0 0 0 0 0 0 0 . X X X X X X X X

In numbers this means: 168.192.0.X
The last byte, designated here as X, may have the values 1 ... 254 (0 must remain available for Ethernet service tools, 255 must remain available for broadcast messages).

A network mask of 0.0.0.0 is not permissible.

When the TCP/IP driver is added to the FST project, the controller gets the ability to communicate via Ethernet. In terms of information technology, we can say that the TCP/IP driver turns the controller into a 'server', that is, into a device that can answer queries. Such a device is called a 'slave' in fieldbus technology.

This does not apply – this is very important – until after the project is downloaded in the controller with the TCP/IP driver. This project – including the TCP/IP driver – can only be loaded via the RS232 interface, as the controller cannot communicate via Ethernet without the TCP/IP driver.

To repeat:
To enable a FST controller to communicate via Ethernet, a project must be downloaded with the TCP/IP driver and IP address with the aid of the serial port to be controlled.

The TCP/IP driver has already been pre-installed at the factory for CPX-FEC.

### 14.1.2 Can the controller be found in the network?

As soon the TCP/IP driver is located in the controller, the controller can be recognised in the network.

A diagnostic tool for the network, the TCP/IP application, is part of the FST software.
You will find this program on your FST CD in the subdirectory 'Tools, TCPIPApplication'. Install this program on your computer and start it.

The TCP/IP application's job is to display all the available controllers in the network. You can recognise each controller with its IP address, hardware address and project name. For all the current CPUs the CPU type and version of the TCP/IP driver is also displayed.

The TCP/IP application provides very fast help for the most frequent errors:

– Addresses that do not match

– Double addressing.

Activating 'Check Alive' will show you the TCPIP application with a green check that communication with a controller is possible – or with the closed sign that communication is not possible.

The TCP/IP application is now looking cyclically for connected devices and displays directly if communication is possible.

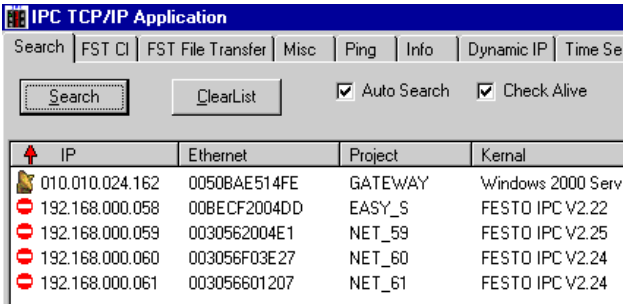



Fig. 14/4

In the window above you can see that the programmer PC on which the TCPIP application is started has the IP address 10.10.24.162, but the connected controllers have addresses from the number range 192.168.0.XXX. Communication between these devices is not possible, as it is forbidden by the network mask 255.255.255.0.

Communication is possible after the programming PC has received another IP address.

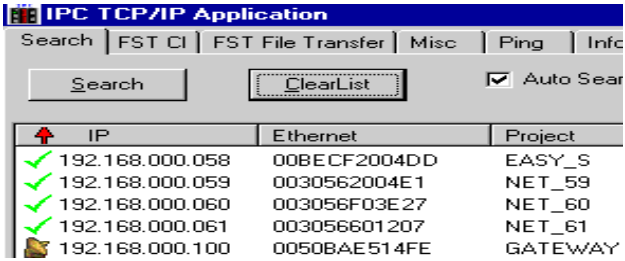



Fig. 14/5

In the following example the IP address is doubled, which makes communication with these devices impossible:

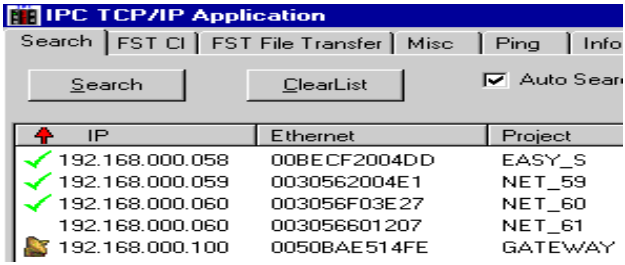



Fig. 14/6

If all the addresses are correctly set up, communication can occur. For each IP address, you can see the Ethernet address (also called a MAC or hardware address), the project name, the kernel version, the TCP/IP driver version and the CPU type [47].

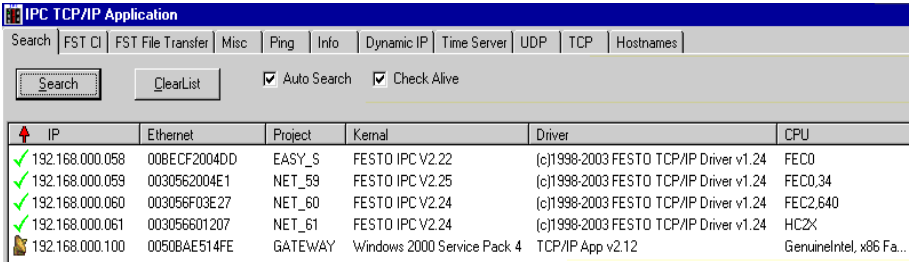



Fig. 14/7

If there are multiple PCs in the network in addition to the controllers, the TCT/IP application will only show these other PCs if the TCT/IP application has been started there as well.

[47] For older CPUs it can be that not all the information can be shown.

### 14.1.3 DHCP server

DHCP servers, which automatically assign an address to new devices, are used in many local networks. The FST TCP/IP driver supports such DHCP servers. To do this the IP address must merely remain 0.0.0.0 in the driver configuration.

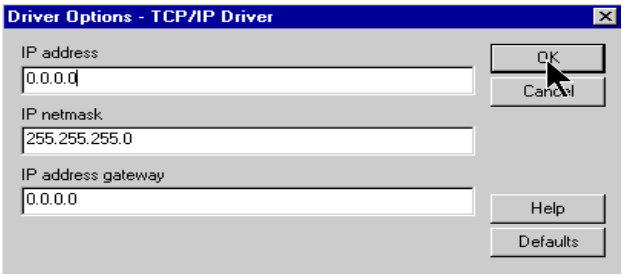



Fig. 14/8

Should the IP address be 0.0.0.0, as in the example above, and a DHCP server be in the network, then the controller will automatically assign an address.

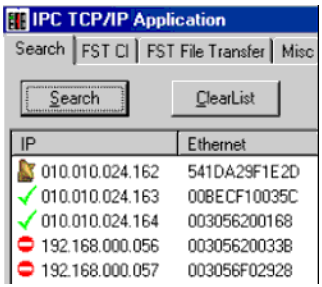



Fig. 14/9

In this example the addresses 10.10.24.XXX are assigned by a DHCP server.

When the DHCP server is used, the programming complexity is somewhat greater for communication between the controllers, as a specific target IP address is required for each send/receive. These IP addresses must be interchangeable at first. If the IP address has been assigned to the controller by the DHCP server, the address must read out in the program and subsequently be broadcasted to all the connected controllers in a previously agreed flag word. Then the target address can then be read out from this flag word for communication.

### 14.1.4 Program sensitive IP addresses

An IP address may need to be set up from outside for some projects. For example, it is conceivable that a machine's IP address should be set up by means of one or more rotary switches. For this purpose a module is available for setting up (and querying) the IP address: IP_IP.

## 14.2 Programming and trouble shooting in the network

Let's assume you have a production line with 10 controllers: Conveyor control, parts supply, part storage, drill, test station, packing station. The controllers can all work away just great. If you want to make a change or search for errors in the system, then take your programming PC and go from controller to controller: Plug – change – test – decouple – next controller – plug – change – test – decouple – next controller ...

It is much easier when the controllers are networked via Ethernet. Each FST CPU is available in a version with an Ethernet connection. Each controller receives an address. And your programming PC is ready to be connected to the network at any spot, so that you can access all the stations. The only thing you need is the wiring, on one hand, and the installation of the TCP/IP driver in the controller, on the other hand.

After the TCP/IP driver has been loaded into the controller using the traditional RS232 interface, the controller has subsequently rebooted automatically, the TCT/IP application has signaled by means of the green check that it is ready for communication, this controller can be addressed with the aid of the IP address in the network.

The connection merely needs to be reconfigured in the FST software. To do this, open the Extras menu and select 'FST Preferences'.

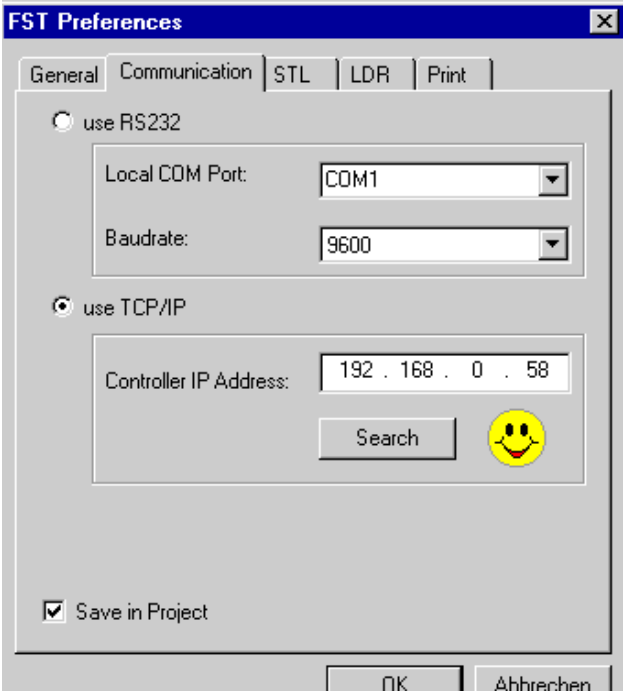



Fig. 14/10

Under the rider Communication you will find the possibility of setting up the Ethernet card and an IP address determined there instead of the RS232 interface.

**Please note**
Please observe:

- TCP/IP use must be selected.
- The IP address must be the address of the controller being communicated with.
- The marking 'Save in Project' means that the setting made here is project sensitive. When a project is re-started, the usual values (mostly RS232 communication) are used at first. Should 'Save in Project' not be selected, this setting will apply to any subsequent new projects.

Clicking OK allows you to communicate with the controller via Ethernet.

Contrary to RS232, Ethernet also provides the possibility of communicating simultaneously with multiple controllers. Simply open FST multiple times, select various projects from the various IP addresses and you can simultaneously observe multiple controllers in different windows.

## 14.3 Communicating among the controllers in the network

2 modules are basically used to communicate among controllers. Easy_R to collect data (receive) and Easy_S to send data. Before these modules can be used, a table or an index list of all the controllers to be communicated with must be compiled using IP_TABLE. As matter of principle, communication takes place in 3 steps:

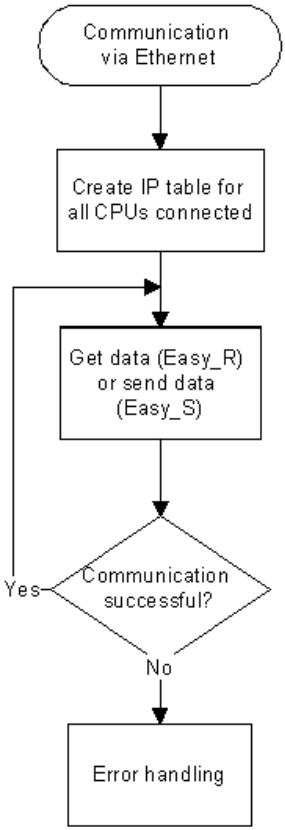



Fig. 14/11

### 14.3.1 IP_Table

A table of all the controllers to be communicated with is compiled using IP_Table (there can be considerably more controllers present in the network; here we are only dealing with the controllers from which data should be collected or to which data should be sent).

| **Index** | **IP address** |
|---|---|
| 1 | 192.168.0.54 |
| 2 | 192.168.0.55 |
| 3 | 192.168.0.56 |
| etc. | |

Tab. 14/1: IP_Table table

As already mentioned above for the DHCP server, the IP address must be specified dynamically at this point when the controllers are newly assigned their IP address by the DHCP server each time after it is switched on. The procedure is as follows:

| **Procedure** |
|---|
| 1. The controller is switched on.<br>2. The controller receives an IP address from the DHCP server.<br>3. The controller detects its own IP address. *)<br>4. The controller broadcasts its own IP address to all the other controllers in the network in a flag field reserved for this controller. *)<br>5. The other controllers use the IP address from this flag field. *) |
| *) To be programmed by the user |

Tab. 14/2

## 14.3.2 Easy_S and Easy_R

An understanding of communication is required to use Easy_S and Easy_R.

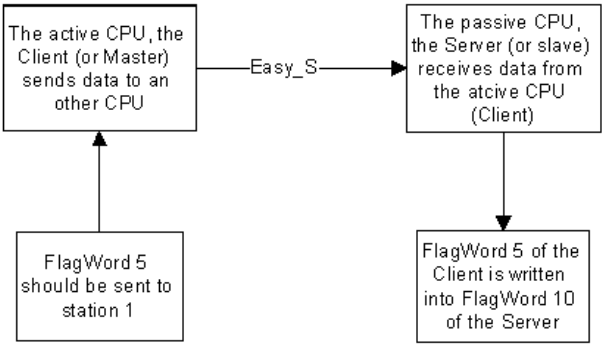



Fig. 14/12

## 14.3.3 Example

The following example [48)] sends the flag word 5 of the client to the flag word 10 of the server. The status of the data transfer is entered in flag word 1 of the client.

[48)] You will find this program example as Easy_S.ZIP on your CD.

```
STEP
"" A table of all the controllers is compiled with IP_Table
"" with which communication should take place.
"" IP_Table must be called up once for each controller.

IF                NOP
 THEN  LOAD       K0
          TO      FW2     'Counter for TimeOut
       CFM 0              " Enter/query IP address in table
         WITH     V1      " 1:enter; 2:query
         WITH     V1      " Index number in IP table
         WITH     V192    " IP address
         WITH     V168    " IP address
         WITH     V0      " IP address
         WITH     V155    " IP address

IF                FU32    'Module parameter
         =        V0
 THEN             NOP

STEP Send
"" Now data is being sent to the controller with index 1
 IF               NOP
 THEN  CFM 2             " Send operand range to controller
         WITH     V1     " Index number in IP table
         WITH     V1     " Operand, 1:M; 2:E; 3:A; 4:R; 11:Strings
         WITH     V1     " Number of operands to be sent (max. 256)
         WITH     V5     " No. of the first operand to be sent
         WITH     V10    " Num. of the first operand in the target CPU
         WITH     V1     " Address of the flag word for the status

STEP Check
"" Now it is checked whether the data transfer was successful or not

"" Data communication was successful
 IF               FW1    'Status of Easy_S
         =        V0
 THEN  LOAD       V0
         TO       FW2    'Counter for TimeOut
       JMP TO Send

"" Data communication produced no answer with the specified time
"" (approx. 50 ms)
 IF               FW1    'Status of Easy_S
         =        V128
 THEN  CV         FW2    'Counter for TimeOut
       JMP TO Send
```

Fig. 14/13

The following image shows the flags for the two participating stations: The client, which sends the flag word 5, is on the top, and the server, which receives the data in flag word 10, is on the bottom.

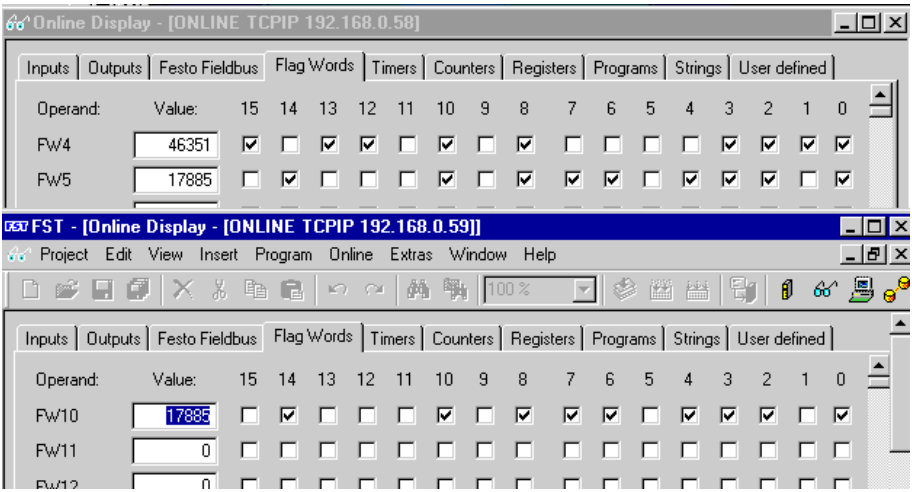



Fig. 14/14

### 14.3.4 Monitoring data communication

Nearly all communication in automation technology must be monitored. The timeouts are simply counted in the above example. Timeouts are perfectly normal in a commensurately large network with a lot of communication. An individual timeout is therefore usually not of any importance. However, if too many timeouts occur, the connection could be completely interrupted. The counter status should thus be evaluated and the proper error message sent out.

In addition, there is also the possibility of testing the principle ability to communicate. A module is available which releases the Ethernet Ping, for example.

## 14.4 Communicating with Windows applications using DDE

Communication between the automation and the office world is of tremendous importance today. Of course, this communication can be custom programmed for the application. The EasyIP protocol for communicating with a FST controller via Ethernet is released just like the CI commands with which communication takes place via the RS232.
Of course, it is easier to use ready-made tools. These include the DDE server and the OPC server.

### 14.4.1 Using DDE to Windows

DDE is a data exchange procedure which Microsoft already introduced with Windows 3.1 and which supports and uses all the subsequent Windows versions. It constitutes a method with which different Windows programs can exchange data with one another. The 'IPC Data Server' is thus a Windows program which makes data available in the format defined by DDE.
All Windows applications understanding this format can now exchange data with the IPC Data Server. The IPC Data Server gets/sends this data from/to connected FST controllers via Ethernet.

The 'IPC Data Server' must be installed separately. You will find this program on your FST CD in the subdirectory Tools. After you have installed the DDE server, you will find the entry IPC Data Server in your program menu. Please start the IPC Data Server.

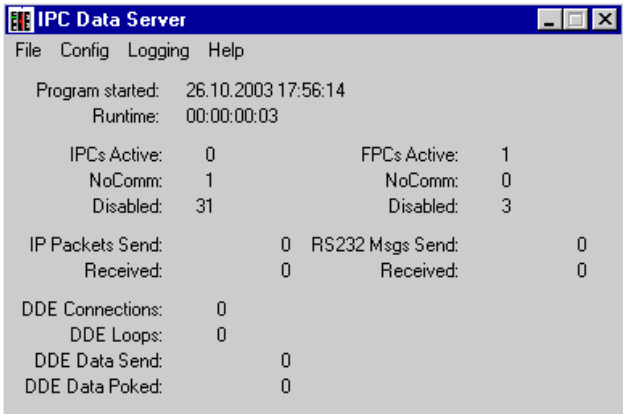



Fig. 14/15

Open the Config. menu.

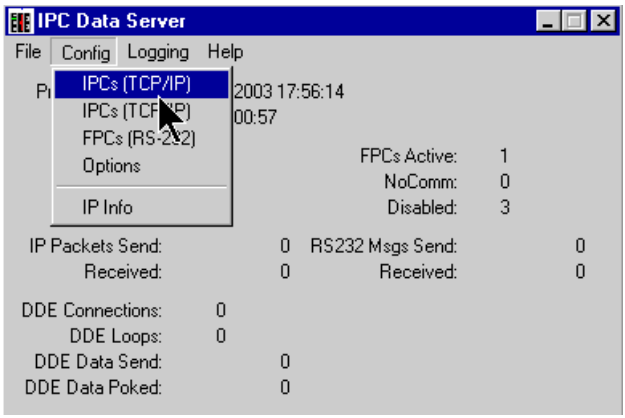



Fig. 14/16

The IPC Data can communicate with connected controllers via both Ethernet and RS232. This applies to all the FST controllers, even older FPC systems (but only via RS232). If the RS232 interface is being used, only one controller per interface can be communicated with. Should your computer be equipped with 4 serial ports, the IPC Data Server can communicate with up to 4 connected controllers via RS232.

The possibilities via Ethernet are more extensive: Communication can take place with up to 48 controllers. Open the configuration for 'IPCs (TCP/IP)'.

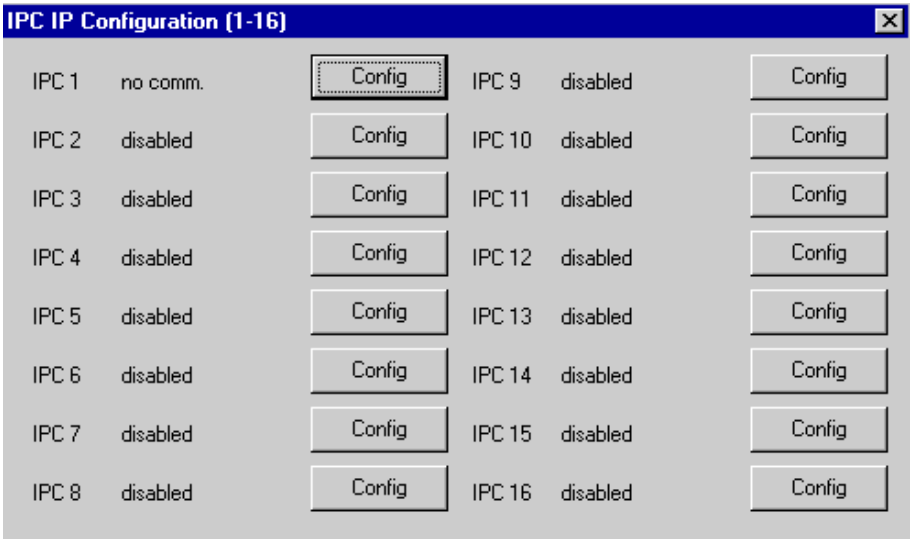



Fig. 14/17

In the first window you will see the access to the first 16 connected systems. Start with the first one – it will be called IPC_1 later.

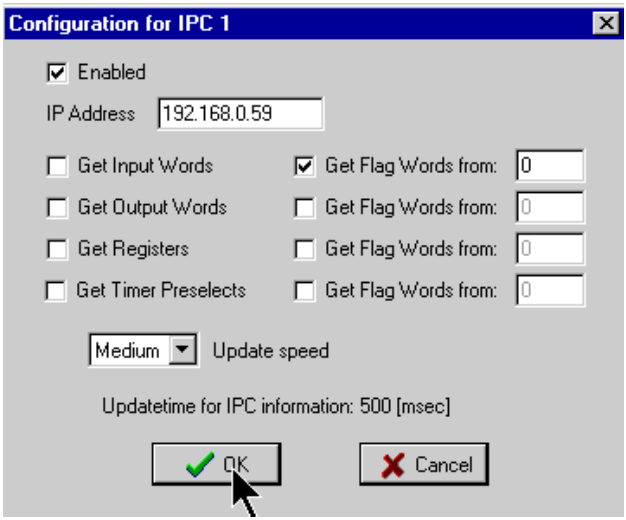



Fig. 14/18

First, communication to this IPC_1 is 'Enabled'.
Afterwards, choose which data should be accessible. Each telegram always transmits 256 words. Click on 'Get Input Words', for example; then all 256 input words are accessible. Click on 'Get Flag Words from 0' (as in the previous example); then the flag words 0 ... 255 are accessible. Altogether the input and output words, the register, the timer preselect and at most 4 x 256 = 1024 flag words from a controller can be made available to a Windows computer.

**Please note**
For use of the IPC Data Server via the serial port it is not necessary to select a certain variable range. You can then directly access any operands via the DDE server.

Should communication with the specified IP address already be possible, the DDE server will be informed of this immediately.


Fig. 14/19

Starting now the data specified in the configuration is available to all DDE-enabled Windows applications. The following linguistic rule applies here:

DDE program name for Ethernet communication:
IPC_DATA
DDE program name for RS232 communication:
FPC_DATA

Controller name for Ethernet communication:
IPC_1 ... IPC_48
Controller name for RS232 communication:
FPC_1 ... FPC_4

Variable name: Operand designations in accordance with FST CI commands must use the original (German) abbreviations:
EW – input word
AW – output word
MW – flag word
TV – timer preselect, etc.

In practice:
If data is being retrieved from the controller to Excel, then proceed as follows:
In the cell in which the data should be seen, the following is entered as a formula:

```
=IPC_DATA|IPC_1!MW5
```

Please observe separation between the application, controller number and the operand designation:

```
=‹applicaiton›|‹controllernumber›!‹operanddesignation›
```

In practice it looks like this:

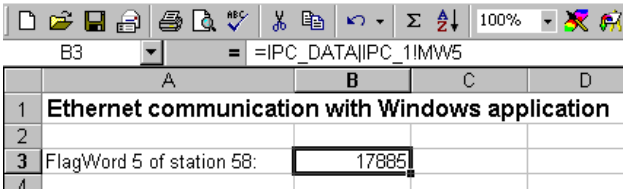



Fig. 14/20

In this procedure data is retrieved from the controller to a Windows application. To send data from a Windows application to the controller, the access from Windows to the DDE data must be programmed. For this, Excel requires a small macro with the following syntax:

```
Sub SendValue()
Channel = Application.DDEInitiate(“ipc_data”, “ipc_1”)
Application.DDEPoke Channel, “MW0”, Sheets(“DDESam-
ple”).Range(“E5”)
Application.DDETerminate Channel
End Sub
```

Applied to the communication with the controller with the IP address 192.168.0.55, configured in the DDE server as IPC_1.

```
Sub SendValue()
Channel = Application.DDEInitiate(“ipc_data”, “ipc_1”)
Application.DDEPoke Channel, “MW5”, Sheets(“Ta-
belle1”).Range(“B2”)
Application.DDETerminate Channel
End Sub
```

Or formulated generally:

```
Sub SendValue()
Channel = Application.DDEInitiate(“‹DDE Program name›”,
“‹Number of the controller›”)
Application.DDEPoke Channel, “‹Operand›”,
Sheets(“‹Name of the spreadsheet›”).Range(“‹Zelle, aus
der die Daten geholt werden (Cells from which data is
retrieved)›”)
Application.DDETerminate Channel
End Sub
```

In doing so the macro 'SendValue' is assigned to a switching element to send to the value because of a request. The result [49] looks like this:

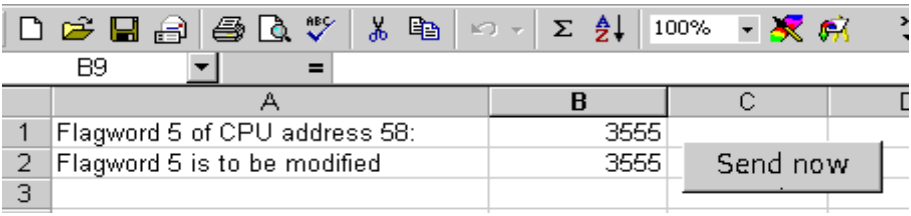



Fig. 14/21

The macro itself looks like this in the macro editor:

```
Sub MW5_Ändern()
Channel = Application.DDEInitiate("ipc_data", "ipc_1")
Application.DDEPoke Channel, "MW5", Sheets("Tabelle1").Range("B2")
Application.DDETerminate Channel
End Sub
```

Fig. 14/22

In the manual for the DDE server you will find more examples, particularly of the usual process visualisations such as Fix, Intouch, Simplicity etc. For VIP, communication via DDE is not necessary but possible, as VIP contains its own driver for FST communication.

[49] You will find this Excel file as DDE-Excel.XLS on your CD.

## 14.5 Some rules for using Ethernet

Ethernet is the most frequently used network across the globe. Ethernet is the network with the most hardware and software support. Ethernet is the network technology with the most experts and the greatest know-how worldwide. However, Ethernet is not yet very widespread in automation technology. That is why it is worth to see what's going on in the other camp and watch the Ethernet experts.

### 14.5.1 Ethernet is standardised

Bus access by Ethernet is standardised in IEE802.3, or rather, the method by which Ethernet positions data on a cable. Ethernet is a Multi Master System, that is, each user is capable of accessing the network sending data. A Multi Master System must resolve the bus access conflict, or rather, clarify what should happen if two or more stations want to send data at the same time. Ethernet resolves this conflict in that each station always also monitors which data is just arriving via the network when it sends data. If multiple stations send at the same time, the signal on the bus is falsified. Therefore, the data to be heard are different from the data which has been sent. This situation is recognised, all the stations cancel the sending process and wait an arbitrarily defined time before attempting to send the data again. This procedure is called CSMA/CD: Carrier Sense Multiple Access/Collision Detection.

This procedures results in the data being able to be sent very quickly, as a station wanting to send data must only wait until the cable is 'free', then it takes place. However, this procedure also results in Ethernet being able to collapse. If too many stations want to send data at the same time, then a collision can come about after the first attempt during the following attempts. At the same time more stations are added which also wish to send data and finally the reply times of Ethernet increase rapidly. The reply time of an Ethernet system increase linearly up to a bus load of 60 %. If this is exceeded, the system will soon come to a stand.

### 14.5.2 Ethernet has standardised cables and plugs

The first Ethernet cables were standardised in 1980. In 1985, the 'thin' coax cable 10Base 2 was standardised and the twisted pair cable 10Base T has been in existence since 1990. This is still the most frequently used wiring method for Ethernet even today.
10Base T is based on the star and a data transfer speed of 10 MBit/s. This means that each station is connected in the shape of a star to a star distributor, the HUB. The decisive advantage of this star topology is that errors can be spotted very easily in the network: Each cable can be individually tested, plugged and removed without the other stations being involved.

Fibre optic cable connections have also been standardised for several years, as have higher data transfer speeds. However, 10 MBit/s is more than satisfactory for the field of control and automation technology and offers greater flexibility in cable length and wiring. Should such a 10 MBit/s network be connected with 100 MBit/s stations, dual-speed HUBs are used so that both data transfer speeds can mix as necessary.

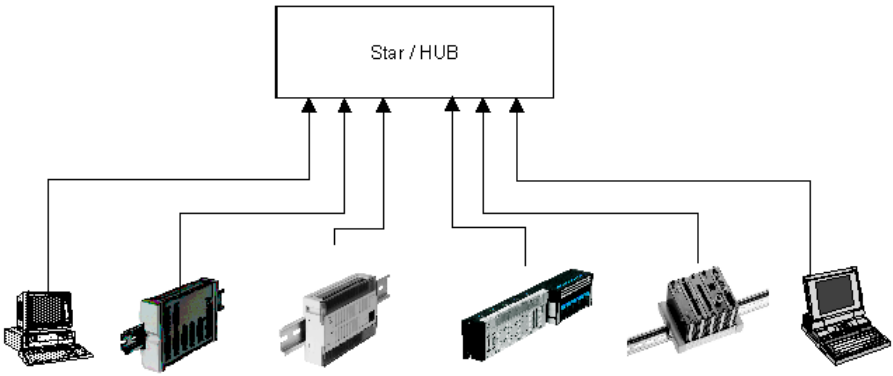



Fig. 14/23

Today hubs are available in designs with up to 128 connections, in both 110 ... 230 V AC and in 24 V DC designs.
Hubs can be cascaded so that systems with several hundred stations are possible.

## 14.5.3 HUBs and switches

A HUB is merely an aid in wiring and trouble shooting. For the rest, all the data is always transmitted via the cables. However, experience has shown (this may not always apply) that particularly in larger networks communication between stations located close to one another is more extensive than between stations separated by great distances.

A switch is just a HUB with the additional characteristic that data arriving on a connection is forwarded to the exact connection bearing the target address of the telegram. Switches can 'learn' the correct connections. The first data telegram is sent to all connections to begin with. The confirmation will reveal which connection is correct. From now on only this one will be served. Subgroups are formed with switches. This usually drastically reduces the total data quantity.

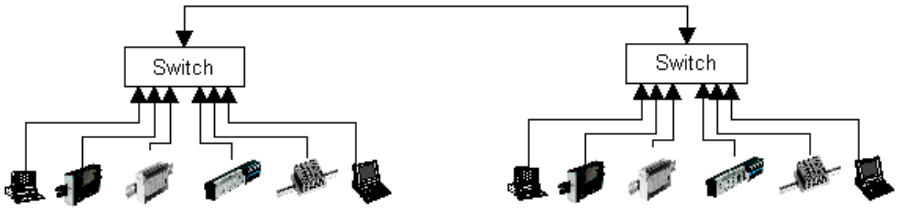



Fig. 14/24

### 14.5.4 From the production line to the office via Ethernet

The connection from production to the office is particularly easy with Ethernet. In doing so, the completely incalculable data quantity in the office comes together with the programmed, predictable data quantity in production. When the office bus load gets to high due to data quantity (considerably more than 30 %), then the CPUs in production are constantly confronted with the task of observing the data traffic in the Ethernet network. This causes the cycle of the program processing to rise. If clearly addressed telegrams are causing the data quantity, conventional switches are enough to effectively protect the office and production from each other. However, should broadcast messages (messages 'to all') sent cyclically by Windows or Novell users be causing the data traffic, then the office and production must be separated using routers or special switches. The router is configured in such a way that targeted telegrams transmitting data between production and the office may pass, but broadcast messages may not. This achieves effective protection between the office and production.

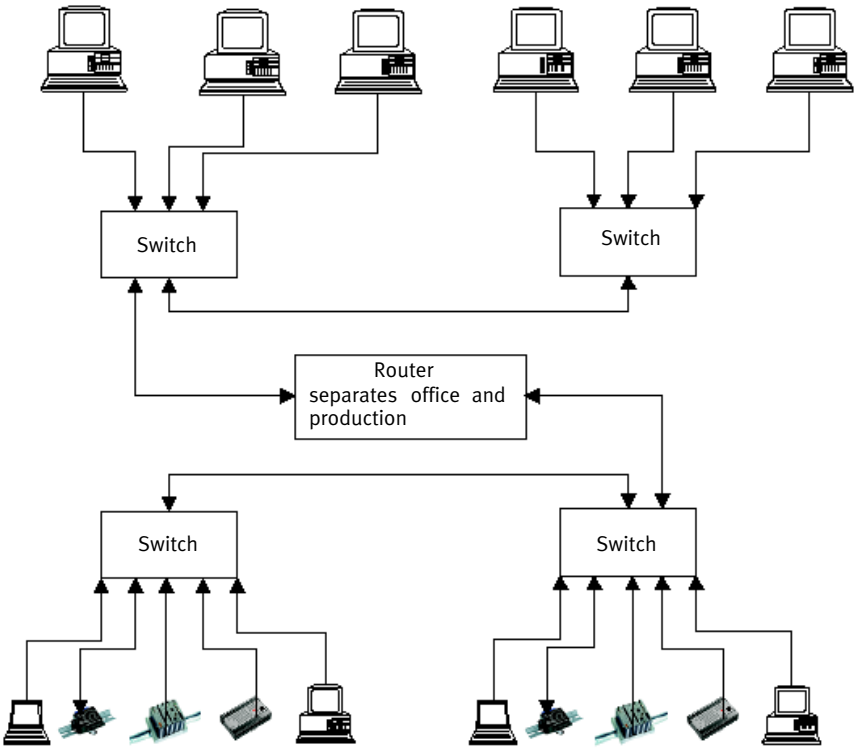

Switch
Switch
Router
separates office and
production
Switch
Switch

Fig. 14/25

### 14.5.5 EasyIP

EasyIP is the name of the protocol with which controllers communicate. EasyIP is ideally suited for communication in automation technology because it

- works completely transparently within local networks, thus is able to work on the same cable as a Windows or Novell network or the like
- uses TCP/IP as its medium of transportation
- is configured in a highly simple manner
- can also be used as a master slave system, if necessary
- can communicate with any Windows application by means of DDE or OPC.

### 14.5.6 The rules for using Ethernet (10 MBit/s)

- For industrial applications which should exchange data under real-time conditions, a bus load of no more than 30 % is recommended (including broadcast telegrams).
- Each individual cable between the HUB and station – so-called patch cable – may be a maximum of 100 m long.
- No more than 3 HUBs may ever be situated between two stations.
- Cables must comply to at least category 5 and be double screened (pairwise and in their entirety).
- The screening must be laid over a large area to ground (e.g with the aid of PS1 ZE30).

# The WEB server in the controller

## Chapter 15

# Contents

## 15.1 What is a WEB server?

A WEB server is a computer that makes data available so that it can be accessed using a browser. There are now millions of servers available on the Internet. For example, you can access the Festo server at http://www.festo.com. If a browser (MS Internet Explorer or Netscape or Opera or ...) contacts this address, data is transferred corresponding to a precisely defined syntax. The browser interprets this syntax and displays the data as specified. The PC, which displays the data with the aid of the browser, is the client wanting to know something from or share something with the server.

The data made available by the server differs greatly. Depending on the origin, the subject matter can be pneumatic cylinders and valve terminals (www.festo.com), Ethernet-enabled controllers, official government publications (www.whitehouse.gov), political propaganda, eroticism ... or production data. If production data is meant, then it could deal with the WEB server in a controller.

## 15.2 WEB Server and FST

Just as with the TCP/IP driver, a driver is also required to use the WEB server in the controller. In addition, the WEB server driver absolutely requires the TCP/IP driver. That means that two drivers are required in the project for this example.

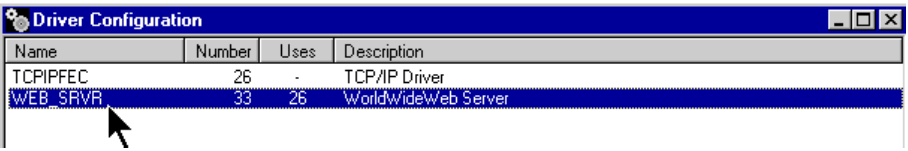



Fig. 15/1

As soon as a project with both of these drivers has been downloaded, the controller can be addressed in the network and observed with a browser (WEB server). Connect the controller and a PC by Ethernet, start the browser on the PC and enter the controller's IP address:


Fig. 15/2

### 15.2.1 Necessary browser settings

Three essential settings are necessary on your browser so that the connection to the controller really works:

– First, the browser must search the network and not just via a connected modem.

  You will find this setting, e.g. for MS Internet Explorer 5 under Tools, Internet Options, Connections.

– Second, the browser may not access directly via a proxy when you are using a direct connection without an intranet.

  You will find this setting, e.g. for MS Internet Explorer 5 under Tools, Internet Options, Connections, Settings.

– Third, it could be necessary to turn off the browser's local clipboard if the controller's data is not refreshed.

  You will find this setting, e.g. for MS Internet Explorer 5 under Tools, Internet Options, General, Temporary Internet Files, Settings.

## 15.2.2 Some rules for the FST WEB server

You should be familiar with some rules when using the FST WEB server:

– As a matter of principle, names with 8+3 characters are necessary; therefore, e.g. index.htm (not: index.html)

– The page index.htm can be part of the FST WEB server if the 'Internal Pages' are activated in the driver configuration (default setting).

– The FST WEB server checks whether a page main.htm exists for each call-up. If the page exists (and a certain page is not explicitly called up), then main.htm is displayed. If main.htm does not exist, then the index.htm, which always exists, is displayed.

## 15.2.3 Where are the HTML pages?

Every FST controller knows drives – just like your PC. The FEC based controllers have A: and B: drives, the A: drive not being able to be changed by the user, and the B: drive being accessible for the project and the application files. All PS1 based CPUs (not the HC0X CPUs, as they correspond in technical terms to the FEC) also know the C: drive, which is mostly used for the application data.

There are two ways of transferring files to the controller. On the one hand, file transfer can be called up from the FST software; on the other hand, files can be transferred using the TCP/IP application.

## Transferring files using the FST software

Use the entry File Transfer in the online menu or the icon for the file transfer.


Fig. 15/3

Then you will see the files in the controller.

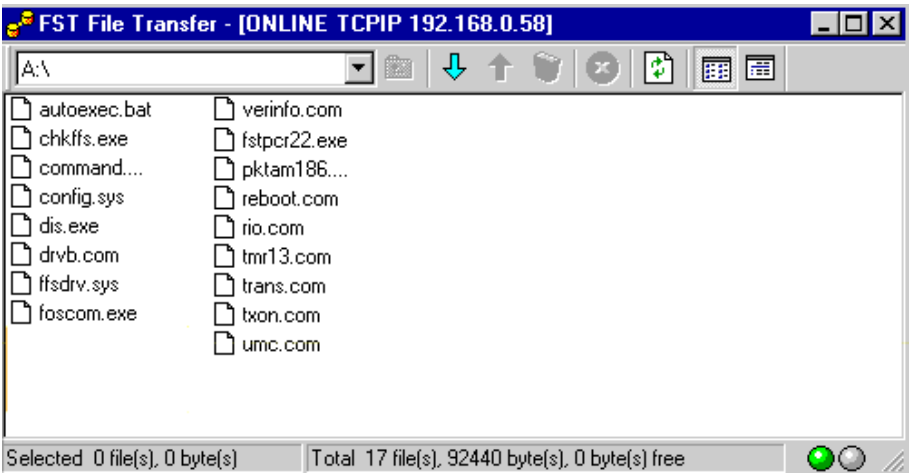



Fig. 15/4

This is drive A: in this example: This drive cannot be changed. After switching to drive B: the project files and drivers can be recognised.

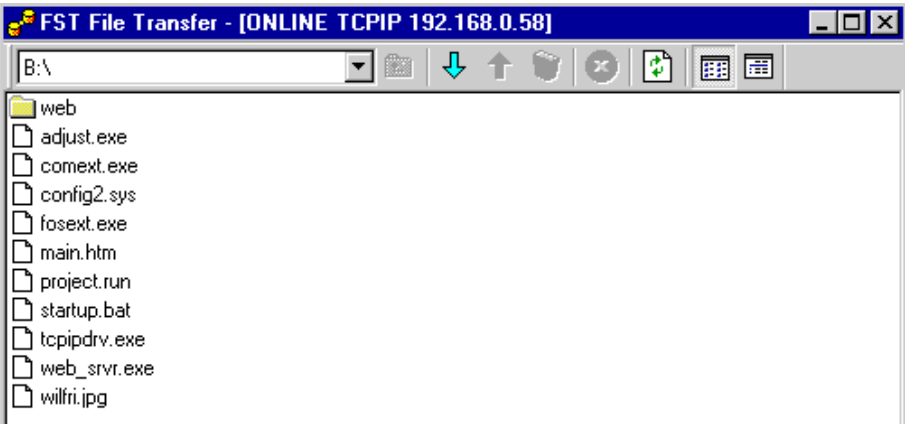



Fig. 15/5

The WEB pages which can be displayed using the browser now also belong in the subdirectory WEB. To transfer files, click the Download arrow.

## Tranferring files using the TCP/IP application

The entry 'FST File Transfer' is in the TCP/IP Application.

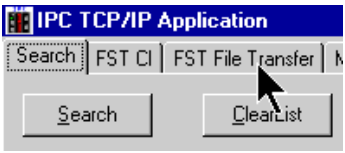



Fig. 15/6

If it is called up, only the correct IP address of the controller needs to be entered to establish the connection.

**Please note**
The IP address can also be copied from the Search window of the TCP/IP application.
Right-clicking the IP address makes a small menu available containing the entry 'Copy IP address'.

After connecting, the controller and the PC can now be seen next to one another.

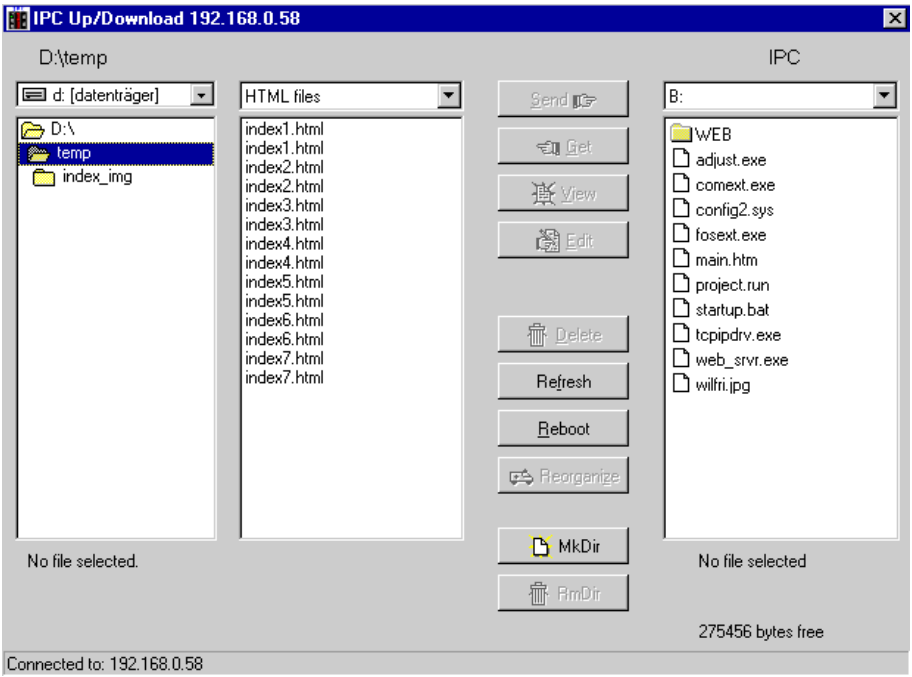



Fig. 15/7

## 15.3 HTML pages for the WEB server

You have already seen above that the WEB server of the FST controller itself already contains data that the browser can display. This start page of the WEB server is normally called INDEX.HTM and can indeed also be called up under this name for FST WEB server.

All the controller's operands can be accessed from this page. The pages integrated in the WEB server are programmed in such a way that they are refreshed cyclically approximately every 5 seconds.

When the variables change, the display will also change.

Even if you may find it quite exciting in the beginning that the WEB server comes with finished WEB pages, the WEB server really only makes sense if you are designing your own pages oriented to your own use.

To make the introduction easier, some basic techniques are described. The easiest place to find instructions for building HTML pages is on the Internet. HTML and the FST controllers are, for example, described at http://fstdemo.beck-ipc.com.

### 15.3.1 The first HTML page

An HTML page is plain text. Any text editor suffices for writing the first page. (Word can only be used if the text is saved as a text file or if Word causes an HTML page to be produced.)

A simple HTML page can be written like this:

```
‹HTML›
‹HEAD›
        ‹TITLE›My first HTML page‹/TITLE›
‹/HEAD›
‹BODY›
        Hello world
‹/BODY›
‹/HTML›
```

Fig. 15/8

Please observe that 'HTML Tags' are always in uppercase/ lowercase characters and are always ended with a preceding /:

```
‹HTML›
‹/HTML›
```

Fig. 15/9

are thus the beginning and end of an HTML page.

If the above HMTL text is saved in the file main.htm, this file is transferred to the B: drive (or the C: drive for PS1 CPUs) and this page is displayed using the browser when the controller is called up.


Fig. 15/10

## 15.3.2 A little bit more text

A little more design makes the page look slightly more interesting:

```
‹HTML›
‹HEAD›
                ‹TITLE›My first HTML page‹/TITLE›
‹/HEAD›
‹BODY›
                ‹H1› This is an example for the FST book ‹/H1› ‹BR›
                ‹H2› This is not an HTML course, ‹BR›
                     it is simply a very small introduction. ‹/H2› ‹BR›

‹/BODY›
‹/HTML›
```

Fig. 15/11

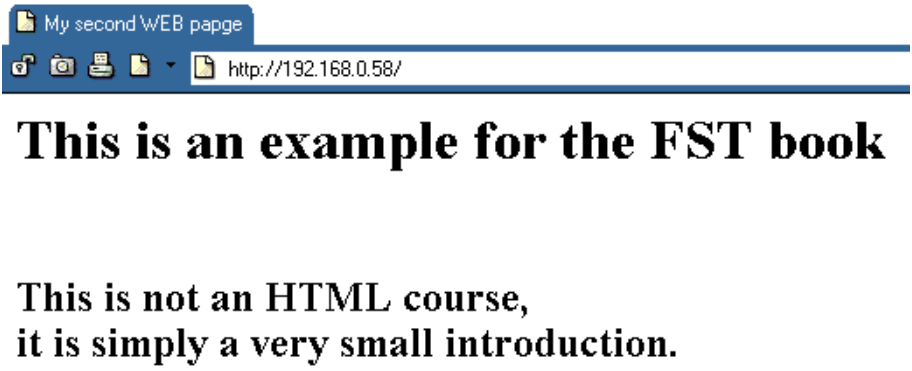



Fig. 15/12

## 15.3.3 Accessing the process

Changing the font size, using graphics or links to other Internet sites are all techniques that are part of all Internet pages and which do not take into account any particularities in automation technology. However, access to the process's variables is decisive for an operating console for a production line. The FST WEB server provides a special HTML tag which speaks the 'controller's language'.

The following is used to read an operand:

```
‹FSTCI DEW0›
```

The command interpreter [50)] of the FST controller is addressed with FSTCI. The display command uses DEW0 for the input word 0 (EW0).

The modify command is used to change an operand. As the change of an operand should not occur statically (then the changed word could be permanently programmed), a link tag is used:

```
‹A href=”main.htm?ci:maw0=255”›Switch the outputs 0 ... 7 to 1‹/A›
```

[50)] In chapter 18 an extremely shortened overview is provided of the CI commands. The CI is described in detail in the FST manual. To address inputs, outputs, flags etc. for this purpose it is necessary to use the original (German) names: E - input, A - output, M - flag etc.

The above example can also be expanded:

```
‹HTML›
‹HEAD›
        ‹TITLE›My first HTML page‹/TITLE›
‹/HEAD›

‹BODY›
        ‹H1› This is an example for the FST book ‹/H1› ‹BR›
        ‹H2› This is not an HTML course, ‹BR›
            it is simply a very small introduction. ‹/H2› ‹BR›
        The input word 0 has the value : ‹FSTCI DEW0› ‹BR›
        ‹A href=”main.htm?ci:maw0=255”›Switch the ouputs 0 ... 7 to 1‹/A›
‹A href=”main.htm?ci:maw0=255”›Switch the ouputs 0 ... 7 to 0‹/A›
‹/BODY›
‹/HTML›
```

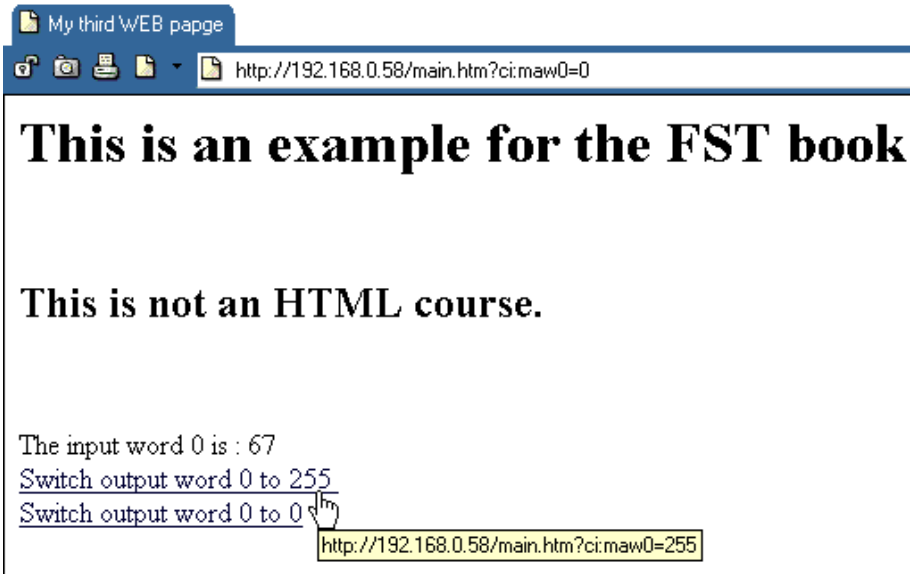



Fig. 15/13

Accessing the process's variables is the basis for visualising and operating a process with the aid of the WEB browser.

# Tips and tricks – Helpful yet fun

## Chapter 16

# Contents

## 16.1 Allocation list

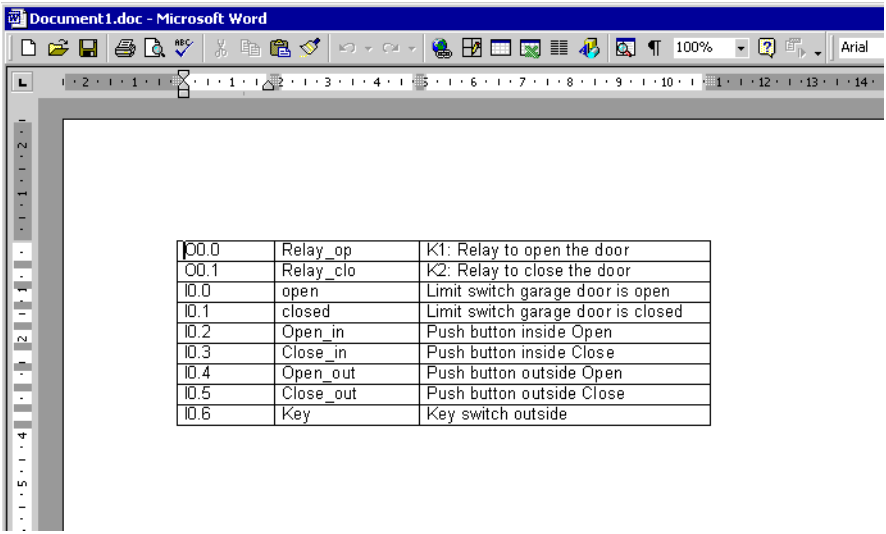
- You can copy the allocation list via the Windows clipboard both from or to Excel. If you would prefer to edit in Excel, then the best way to start is with an entry in the allocation list, which you then copy to Excel. There you can edit as you please, mark the entire list and copy it back to the allocation list editor.

E6 =

| | A | B | C |
|---|---|---|---|
| 1 | O0.0 | Aplus | Y1: Clamp the workpiece |
| 2 | O0.1 | Aminus | Y2: Release the workpiece |
| 3 | O0.2 | Driller | Y3: Drilling cylinder |
| 4 | O0.3 | Ejector | Y3: Eject the workpiece |
| 5 | O0.4 | L_Start | H1: Lamp Start |
| 6 | O0.5 | L_Home | H2: Lamp Home Position |
| 7 | O0.6 | L_STOP | H3: Lamp Emergency Stop |
| 8 | O0.7 | L_Auto | H4: Lamp Automatic Mode |
| 9 | I0.0 | Released | B1: Clamp cylinder is released |
| 10 | I0.1 | Clamped | B2: Clamp cylinder has clamped |
| 11 | I0.2 | Drill_dow | B3: Drill cylinder is down |
| 12 | I0.3 | Drill_up | B4: Drill cylinder is up |
| 13 | I0.4 | Ejected | B5: Ejector is forward |
| 14 | I0.5 | Ej_back | B6: Eject cylinder is back |
| 15 | I0.6 | Start | S1: Push button Start |

Fig. 16/1

- If you wish, you can also create the allocation list in Word. For this, create a table in Word. Mark the entire table and copy it to FST.




| | | |
|---|---|---|
| Q0.0 | Relay_op | K1: Relay to open the door |
| Q0.1 | Relay_clo | K2: Relay to close the door |
| I0.0 | open | Limit switch garage door is open |
| I0.1 | closed | Limit switch garage door is closed |
| I0.2 | Open_in | Push button inside Open |
| I0.3 | Close_in | Push button inside Close |
| I0.4 | Open_out | Push button outside Open |
| I0.5 | Close_out | Push button outside Close |
| I0.6 | Key | Key switch outside |

Fig. 16/2

- You can sort the allocation list as you please. It is sorted by default in alphabetical and numeric order by operand address. That is why I0.0 is the first entry. Click – as you do in Windows – the column icon., e.g. symbol, and the allocation list is sorted according to the symbol name.

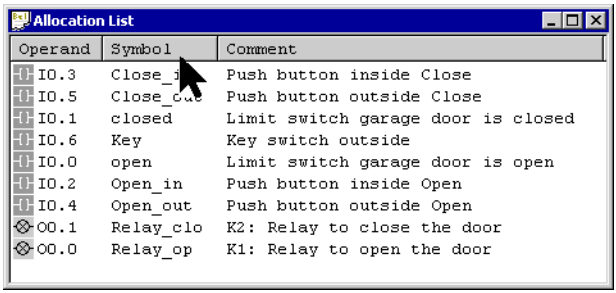



Fig. 16/3

## 16.2 Statement list

- The FST statement list has a clear priority: AND comes before OR. In a mixed logic AND will thus always be linked first, then OR. The garage door example could thus also be programmed without brackets.

```
""Open garage doors
IF             Open_in       'I0.2: Push button inside Open
      OR       Open_out      'I0.2: Push button outside Open
      AND      Key           'I0.6: Key switch outside
      AND  N   Relay_clo     'O0.1: K2: Close garage door
      AND  N   Close_in      'I0.3: Push button inside Close
      AND  N   Close_outs    'I0.5: Push button outside Close
      AND  N   open          'I0.0: Limit keys garage door is open
THEN  SET      Relay_op      'O0.0: K1: Open garage door
```

Fig. 16/4

## 16.3 Updating the project

FST allows entire projects to be downloaded [51]. However, as long as the project changes contain no new modules or the like, a 'Update Project' is possible. You will find this entry in the Online menu. In Update Project the project changes are analysed and a comparably small file is transferred containing just these changes. The changes are then integrated into the project in the running CPU operation.

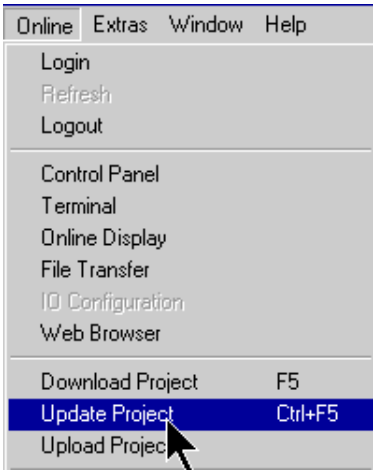



Fig. 16/5

[51] Old 'FST pros' may remember that you could also download individual programs in FST. This is not possible anymore.

## 16.4 Finding syntax errors

If there is an error in your project, FST will not completely compile the project and will also not transfer it to the controller. The message window shows which error has been found and where it is suspected to be. The following example shows such an error.

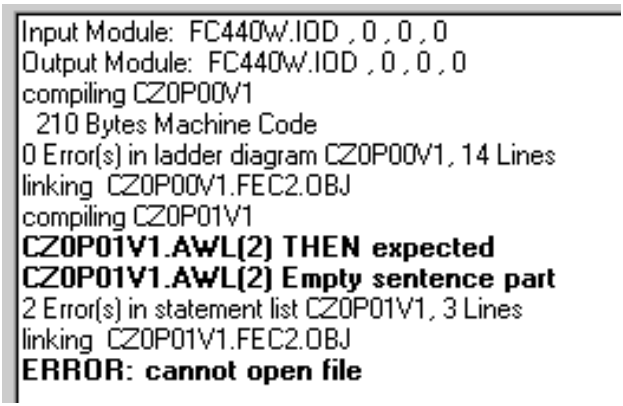



Fig. 16/6

The line

```
CZ0P00V1.AWL(5) Invalid sequence of setence
part
```

points out where the error can be found:

```
CZ0POOV1.AWL(5) Invalid sequence of sentence part
   |   |     |  +—— Explanation of the error
   |   |     +————— Program lines which the error is suspected
   |   +———————————— Version 1
   +———————————————— Program 0
```

Double-clicking the error location (here CZ0P00V1.AWL(5)) leads directly to the program.

# Error messages

## Chapter 17

# Contents

## 17.1 General FST error numbers

| Error number | Meaning |
|---|---|
| 0 | No error |
| 2 | Checksum error in project file PROJECT.RUN.<br>• Download complete project again |
| 6 | Program 0 must be started but is not available.<br>• Create and download Program 0. |
| 7 | Attempt to set or restore a non-existing program or its status.<br>• Change the program or download the missing program. |
| 9 | The program cannot be started due to errors in the project file PROJECT.RUN.<br>• Check the project and download it again. A driver could possibly be used but it is not loaded, or a driver is loaded but cannot be used because the conditions for it are not met. The hardware needed for the driver may not be available or is not configured properly. The controller may not have enough memory. |
| 11 | I/O card defect, short circuit at the output or no power supply.<br>• Replace I/O card, clear short circuit or connect power supply. |
| 12 | I/O card cannot be found.<br>• Check I/O card.<br>• Check the switch setting on the card and in the I/O configuration. |
| 13 | Watchdog expired<br>A driver, module or IO script blocked the run-time system for more than 1 second and triggered a restart. |
| 14 | The driver to be started cannot be found. A required driver cannot be found or is not executable because of an initialisation error.<br>The environment (hardware, parameters) might not be available correctly for the driver.<br>• Integrate the driver, correctly set the parameters in the driver configuration and check the hardware. |
| 36 | Nested CMP/CFM or CMP/CFM cannot be found when calling up.<br>Change program structure |
| 39 | Double error, error in program.<br>• Eliminate source of error. |
| 42 | CPX diagnosis<br>• Clear CPX error |

| Error number | Meaning |
|---|---|
| 57 | The project file (PROJECT.RUN) cannot be read.<br>• Download project again. |
| 59 | Arithmetic error.<br>• Modify program. |

Tab. 17/1

## 17.2 Festo fieldbus error

| Error number | Meaning |
|---|---|
| 14 | Critical driver error<br>Project cannot be started when there is a “soft” error reaction.<br>This error occurs when the fieldbus driver could not be found or if problems occur when starting the firmware of the CP61. |
| 60 | Actual configuration is not a superset of the set configuration.<br>• Correct the fieldbus configuration or connect any missing users |

Tab. 17/2

## 17.3 AS-Interface error

| Error number | Meaning |
|---|---|
| 700 | No configuration data available for the ASi driver<br>• Create configuration file |
| 701 | Master not available<br>• Check the rotary switch setting of the master. Check configuration:<br>For the HC20 only the master with switch setting 1 or 4 can be used, for HC16 setting 1 |
| 702 | Master not available<br>• Check the rotary switch setting of the master. Check configuration:<br>For the HC20 only the master with switch setting 1 or 4 can be used, for HC16 setting 2 |
| 703 | Master not available<br>• Check the rotary switch setting of the master. Check configuration:<br>For the HC20 only the master with switch setting 1 or 4 can be used, for HC16 setting 3 |
| 704 | Master not available<br>• Check the rotary switch setting of the master. Check configuration:<br>For the HC20 only the master with switch setting 1 or 4 can be used, for HC16 setting 4 |
| 711 | Failure of a slave<br>• Check the slave, replace it, check the configuration, address 1 |
| 712 | Failure of a slave<br>• Check the slave, replace it, check the configuration, address 2 |
| 713 | Failure of a slave<br>• Check the slave, replace it, check the configuration, address 3 |
| 714 | Failure of a slave<br>• Check the slave, replace it, check the configuration, address 4 |

Tab. 17/3

## 17.4 PROFIBUS-DP (CP62) error

| Error number | Meaning |
|---|---|
| 1001 | The driver cannot make a connection to the card.<br>Probable causes:<br>– The card address does not agree with the value for the driver configuration.<br>– No valid configuration data for the card (SyCon) |
| 1004 | Configured slave not on the bus (only in the event of a “hard reaction”) |
| 1005 | Input address range exceeded (offset) |
| 1006 | Output address range exceeded (offset) |

Tab. 17/4

# The FST CI command interpreter

## Chapter 18

# Contents

A FST controller communicates with the programming PC or with any other device connected to the programming interface with the aid of a clearly defined, very simple command code, which is designated by the FST CI (Command Interpreter). You will find details on the FST CI in the FST manual. An overview of the most important CI commands should suffice here.

## 18.1 The FST operands / variables

The most important operands are:

| Operand designation | Address range | Example bit addressing | Example word addressing |
|---|---|---|---|
| Inputs | E0.0 ... E255.15 | E0.12 | EW147 |
| Outputs | A0.0 ... A255.15 | A47.3 | AW0 |
| Flag | M0.0 ... M9999.15 | M4312.14 | MW9999 |
| Register | R0 ... R255 | – | R36 |
| Programs | P0 ... P63 | P14 | – |
| Program status | PS0 ... PS63 | PS14 | – |
| Program modules | BAP0 ... BAP99 | BAP12 | – |
| Function modules | BAF0 ... BAF99 | BAF99 | – |
| Timer status | T0 ... T255 | T14 | |
| Timer preselect (set value) | TV0 ... TV255 | – | TV14 |
| Timer word (actual value) | TW0 ... TW255 | | TW14 |
| Counter status | Z0 ... Z255 | Z33 | – |
| Counter preselect (set word) | ZV0 ... ZV255 | – | ZV33 |
| Counter word (actual value) | ZW0 ... ZW255 | – | ZW33 |

| Operand designation | Address range | Example bit addressing | Example word addressing |
|---|---|---|---|
| Error word | F | – | F |
| Constants | K0 ... K65535<br>K$0 ... K$FFFF<br>K-32767 ... +32767 | – | K10 |

Tab. 18/1

### 18.1.1 Displaying operands

The Display command displays operands. Example:

| | |
|---|---|
| DEW0 | Display input word 0 |
| DR5 | Display register 5 |

### 18.1.2 Modifying operands

The Modify command modifies operands. Example:

| | |
|---|---|
| MAW4=33 | Modify output word 4 to the value 33 |
| MM300.14=1 | Modify flag 300.14 to the value 1 |

## 18.2 Some other commands

| Meaning | Example |
|---|---|
| Call up FST CI | ‹CTRL›T |
| Start project – RUN | R |
| Stop project – Stop | S |
| Start a certain program | RP1 |

Tab. 18/2

# Conditions of use for ”Electronic documentation”

**I. Protection rights and scope of use**
The file of your choice is subject to safeguarding provisions. Festo or third parties have protection rights for this electronic documentation which Festo provides on portable data storage devices (diskettes, CD ROM, cartridge discs), as well as in Internet and/or Intranet, always referred to in the following as “electronic documentation”. In so far as third parties have whole or partial right of access to this electronic documentation, Festo has the appropriate rights of use. Festo permits the user the use under the following conditions:

**1. Scope of use**
a) The user of the electronic documentation is allowed to use this documentation for his own, exclusively company-internal purposes on any number of machines within his business premises (location). This right of use includes exclusively the right to save the electronic documentation on the central processors (machines) used at the location.
b) The electronic documentation may be printed out on a printer at the location of the user as often as desired, providing this printout is printed with or kept in a safe place together with these conditions of use and other user instructions.
c) With the exception of the Festo Logo, the user has the right to use pictures and texts from the electronic documentation for creating his own machine and system documentation. The use of the Festo logo requires written consent from Festo. The user himself is responsible for ensuring that the pictures and texts used match the machine/system or the relevant product.
d) Further uses are permitted within the following framework:
Copying exclusively for use within the framework of machine and system documentation from electronic documents of all documented supplier components.
Demonstrating to third parties exclusively under guarantee that no data material is stored wholly or partly in other networks or other data storage devices or can be reproduced there.
Passing on printouts to third parties not covered by the regulation in item 3, as well as any processing or other use, is not permitted.

**2. Copyright note**
Every ”Electronic document” receives a copyright note. This note must be included in every copy and in every printout.
Example: © 2003, Festo AG & Co. KG,
D-73726 Esslingen, Germany

**3. Transferring the authorization of use**
The user can transfer his authorization of use in the scope of and with the limitations of the conditions in accordance with items 1 and 2 completely to a third party. The third party must be made explicitly aware of these conditions of use.

**II. Exporting the electronic documentation**
When exporting the electronic documentation, the licence holder must observe the export regulations of the exporting country and those of the purchasing country.

**III. Guarantee**
1. Festo products are being further developed with regard to hardware and software. The hardware status and, where applicable, the software status of the product can be found on the type plate of the product. If the electronic documentation, in whatever form, is not supplied with the product, i.e. is not supplied on a data storage device (diskette, CD ROM, cartridge disc) as a delivery unit with the relevant product, Festo does not guarantee that the electronic documentation corresponds to every hardware and software status of the product. In this case, the printed documentation from Festo accompanying the product is alone decisive for ensuring that the hardware and software status of the product matches that of the electronic documentation.
2. The information contained in this electronic documentation can be amended by Festo without prior notice and does not commit Festo in any way.

**IV. Liability/limitations of liability**
1. Festo provides this electronic documentation in order to assist the user in creating his machine and system documentation. In the case of electronic documentation which in the form of portable data storage devices (diskettes, CD ROM, cartridge discs) does not accompany a product, i.e. which are not supplied together with that product, Festo does not guarantee that the electronic documentation separately available / supplied matches the product actually used by the user. The latter applies particularly to extracts of the documents for the user’s own documentation.
The guarantee and liability for separately available / supplied portable data storage devices, i.e. with the exception of the electronic documentation provided in Internet/Intranet, is limited exclusively to proper duplication of the software, whereby Festo guarantees that in each case the relevant portable data storage device or software contains the latest status of the documentation. In respect of the electronic documentation in Internet/Intranet it is not guaranteed that this has the same version status as the last printed edition.
2. Furthermore, Festo cannot be held liable for the lack of economic success or for damage or claims by third parties resulting from the use of the documentation by the user, with the exception of claims arising from infringement of the protection rights of third parties concerning the use of the electronic documentation.
3. The limitations of liability as per paragraphs 1 and 2 do not apply if, in cases of intent or wanton negligence or the lack of warranted quality, liability is absolutely necessary. In such a case, the liability of Festo is limited to the damage recognizable by Festo when the concrete circumstances are made known.

**VI. Safety guidelines/documentation**
Guarantee and liability claims in conformity with the regulations mentioned above (items III. and IV) can only be made if the user has observed the safety guidelines of the documentation in conjunction with the use of the machine and its safety guidelines. The user himself is responsible for ensuring that the electronic documentation, which is not supplied with the product, matches the product actually used by the user.