The FST PLC Operating System (Kernel)

In order to use an IPC as PLC, the necessary PLC operating system software must be loaded into the IPC. For this, FST uses the runtime kernel FSTPCR22.EXE. This contains PLC properties that are always required. The kernel is also responsible for loading and executing the user project.

For the controllers that are prepared for FST, the FST PLC operating system is automatically started on power-up.

Additional parts of the PLC operating system can be subsequently loaded into the controller as modules and/or as drivers together with a user project.

The properties of the FST PLC operating system are described in the following sections.

PLC operands

PLC programs consist of program code for handling of data. This data is available to the PLC in the form of operands. It contains not only the values of operands, but also flags and many special operands with specific properties. These operands are the classical PLC operands.

Operands with a value of one bit only are differentiated from those with 16 bits, called words. In some cases, the operands can be used either as a bit value or together in the form of a word value. In the case of FST software this applies to all inputs, all outputs, and all flags. All other operands can only be used either as word or only as bit.

Where operands can be accessed either as bit or jointly in the form of a word, the convention is that the least significant bit (extreme right) contains the bit with the number 0. It therefore follows that the most significant bit (extreme left) is bit number 15.

Other operands have special properties defined by the operating system. For example all counters can be used either in the form of a counter value as counter word CW and in the form of a 1-bit counter status C (expired or not expired). Timers are a similar case, having a timer word TW and a timer status T. The FST PLC operating system ensures that these operands always change their values accordingly.

The individual operand types and their application in programs are described in the following.

Abbreviations

```
        b - Bit number
        w - Word number
```

Value ranges of multibit operands: All multibit operands are 16-bit values.

```
        0 to 65535              (Unsigned decimal)
        -32768 to +32767        (Signed decimal)
        $0000 to $FFFF          (Hexadecimal)
```

Inputs

256 possible input words (0 to 255) each with 16 bits (0 to 15), can be addressed as words and as bits.

```
        Input (bit):
        Syntax:         Iw.b
        Operations:     Interrogate

        Input word:
        Syntax:         IWw
        Operations:     Compare
```

Outputs

256 possible output words (0 to 255) each with 16 bits (0 to 15), can be addressed as words and as bits.

```
        Output (bit):
        Syntax:         Ow.b
        Operations:     Interrogate, set, reset,
                        assign, assign negated

        Output word:
        Syntax:         OWw
        Operations:     Load, compare
```

Flags

10,000 flag words (0 to 9999) each with 16 bits (0 to 15), can be addressed as words and as bits.

```
        Flag (bit):
        Syntax:         Fw.b
        Operations:     Interrogate, set, reset,
                        assign, assign negated

        Flag word:
```

```
Syntax:         FWw
Operations:     Load, compare
```

Registers

256 registers (0 to 255), can only be addressed as words.

```
Syntax:         Rw
Operations:     Load, compare
```

Timers

256 timers (0 to 255), can be programmed either as pulse (T), switch-on delay (TON) or switch-off delay (TOFF) timers in LDR; in STL only pulse timers are available.

```
Timer status (bit):
Syntax:         Tnn, TONnn, TOFFnn
Operations:     Interrogate, set, reset, assign,
                assign negated

Timer value:    TWnn
Operations:     Load, compare

Timer preset:   TPnn
Operations:     Load, compare
```

The type of timer is defined simply by the designation of the timer operand (nn represents the address of the timer):

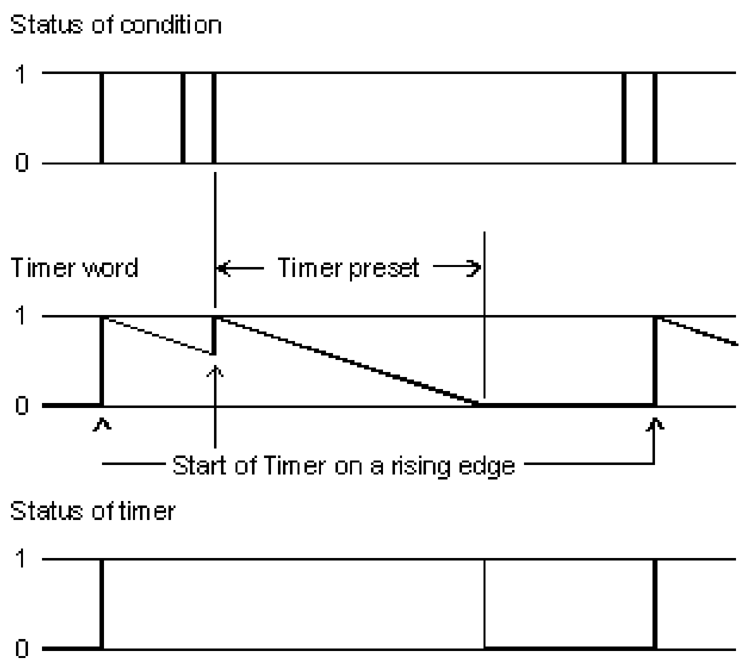
```
Pulse timer             =       Tnn
Switch-on delay timer   =       TONnn
Switch-off delay timer  =       TOFFnn
```

Note! The timer type for a timer should remain the same. A switch-off delay timer should not be used elsewhere as a different type of timer, for example as a pulse timer.

Pulse timer

A pulse timer allows an output to be activated for a pre-determined time by an input pulse. A pulse timer responds only to a rising edge of the input condition (pulse). This occurs when there is a change in status from 0 to 1. The pulse starts the timer (Tnn=1). The timer preset (start value) is loaded into the timer word and the timer starts running. The timer word is then decremented until:

- It reaches the value 0, the timer has then expired (Tnn=0, TWnn=0)
- Another rising edge (pulse) is detected in the conditional part, as a result of which the timer is started again (timer restart)
- The timer status is reset (Tnn=0, Twnn=0)



Example in LDR:

```
¦
¦ I0.1     T1                                    +  T1  +
+--] [--+--]/[--+-------+-------+-------+-------+-¦  10s ¦
¦                                                ¦ Timer¦
¦                                                +------+
¦  T1                                              O0.1
+--] [--+-------+-------+-------+-------+-------+--(   )--¦
¦
```

A pulse at input I0.1 activates the output for the length of time determined by the timer preset (10 seconds). The normally closed contact in the first rung prevents the timer from being restarted if it is already active.

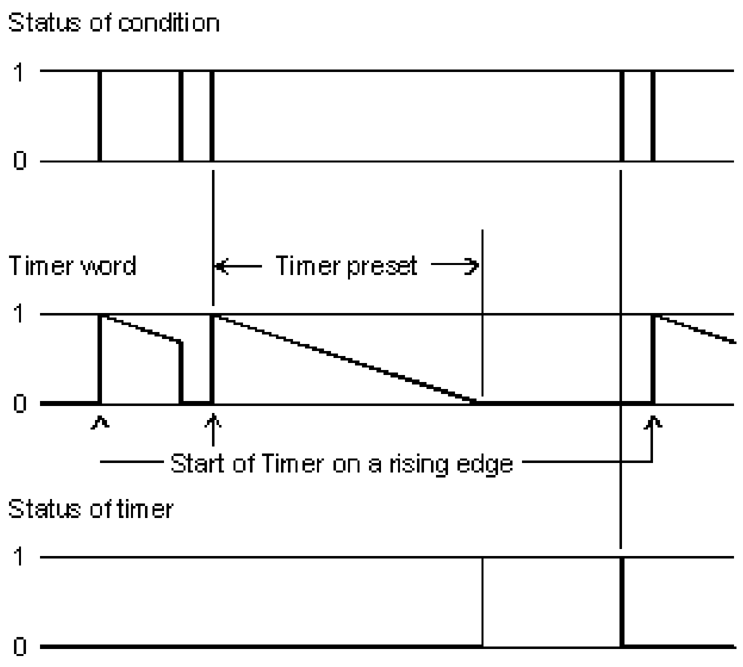
Example in STL:

```
IF                      I0.1
        AND     N       T1
THEN    SET             T1
        WITH            10S
IF                      T1
THEN    SET             O0.1
OTHRW   RESET           O0.1
```

Switch-on delay timer

This timer allows outputs to be activated with a 1-signal after a delay time has expired. The timer preset represents the delay period. When the timer is started, the timer status TON does not become 1 until after the delay time has expired. A rising edge in the conditional part starts the timer. It starts running until:

- The timer word reaches the value 0, the timer has then expired (TONnn=1, TWnn=0)
- The condition changes to a 0-signal



Example in LDR:

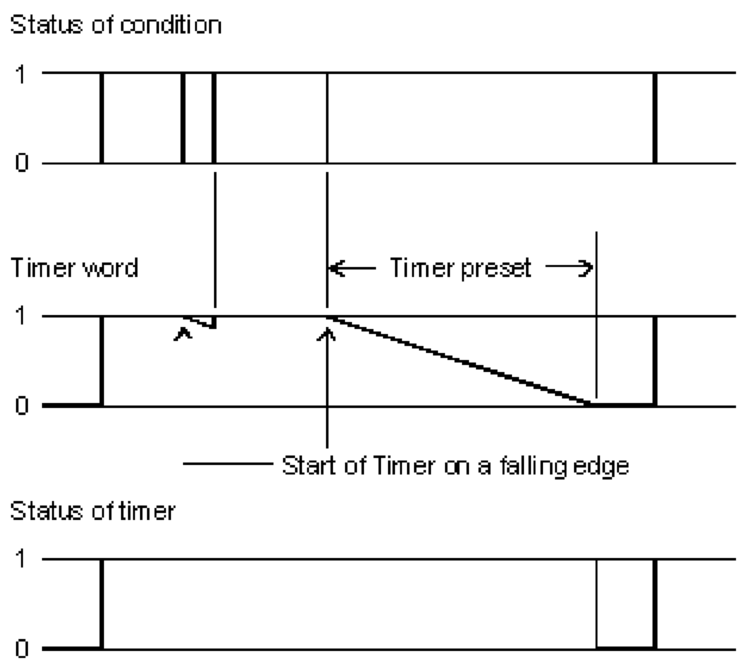
```
¦
¦ I0.2                                            + TON2 +
+--] [--+-------+-------+-------+-------+-------+-¦  20s ¦
¦                                                 ¦ Timer¦
¦                                                 +------+
¦  TON2                                             O0.1
+--] [--+-------+-------+-------+-------+-------+--(    )--¦
¦
```

If a 1-signal is present at I0.2, the output is not activated until after the timer has expired. It remains active for as long as the 1-signal is present at I0.2.

Switch-off delay timer

This timer allows outputs to be deactivated with a 0-signal after a delay time has expired. When a rising edge is recognised for the condition, the timer preset is loaded into the timer word. A trailing edge starts the timer, which runs until:

- The timer word has reached the value 0, the timer has then expired (TOFFnn=0, TWnn=0)
- The timer is reinitialised by a rising edge for the condition



Example in LDR:

```
¦
¦ I0.3                                             + TOFF3+
+--] [--+-------+-------+-------+-------+-------+-¦  30s ¦
¦                                                  ¦ Timer¦
¦                                                  +------+
¦  TOFF3                                              O0.1
+--] [--+-------+-------+-------+-------+-------+--(    )--¦
¦
```

The timer is started by a trailing edge at I0.3. TOFF3 and therefore also output O0.1 continue to carry a 1-signal until the timer has expired.

Counters

```
Operations:     One-bit: Interrogate,
                set, reset, assign, assign negated
                Multibit: Load, compare
C0 to C255      Counter status, one-bit
CW0 to CW255    Counter word, multibit
CP0 to CP255    Counter preset, multibit
```

Constants

```
Operations:     Load, compare
Vnnnn           Multibit
```

Function units

256 function units, 7 of which (32 to 38) are used for passing parameters, the rest are available for general use.

```
Operations:     Load, compare
FU0 to FU255    Multibit
```

Function units FU32 to FU38 are available for each program because they are used for passing parameters to modules. Only the program’s own FUs can be accessed from within the programs, the syntax is therefore simply FUw. The FUs of all programs can be addressed in the command interpreter, for which the program number has to be specified: FUp.w (p = program number, w = FU number 32 to 37).

Programs

Operations:	Interrogate, set, reset
P0 to P63	One-bit

Program statuses

Operations:	Interrogate, set, reset, assign, assign negated
PS0 to PS63	One-bit

The program and program status operands provide information about the status of a program:

P	PS	Status
0	0	Inactive
0	1	No meaning
1	0	Active but suspended
1	1	Active and being processed

A program with the number x can be started and stopped via the operand Px. After having been started with the command SET Px, Px=1 and PSx=1. When it is stopped with the command RESET Px, Px=0 and PSx=0. The active program Px can be suspended with RESET PSx.

Errors

Operations:	Interrogate, reset
E	One-bit
EW	Multibit

Initial execution flags

When a program is executed for the first time the initial execution flag is 1, then it is set automatically to 0.

Operations:	Interrogate
FI	One-bit, for each program

There are separate initial execution flags for each program. Addressing is as for function units.

Retentive operands

If operands are retentive this means that their value is preserved during power off.

On the different controller types different parts of the FST operands are retentive.

```
FECs    A drop in operating voltage results in some of the operands
        being copied to flash disk:

        FW0 to 255
        R0 to 127
        TP0 to 127
        C0 to 127
        CP0 to 127
        Password

HCXX    Part of the zero-power RAM is used for retentive storage of operands.
        As a result, nearly all operands can be made retentive:

        FW
        R
        TV
        C, CP, CW
        FU
        Password
```

Configuration of the driver for retentive operands

For the FEC Compact, FEC Standard, HC0X and HC20 CPUs the above-mentioned operands are always retentive. It is not necessary to configure any driver or something else. The retentive storage of the operands cannot be switched off. Use the function module F9 (see below) to reset all operands if required.

In order to make the above-mentioned operands retentive in the HC1X CPUs, the driver for retentive operands "DRAD" must be selected and parameterised in the driver configuration of each project that wants to use retentive operands. For "Special parameters", the default setting "-q" should normally remain. This option suppresses multiline output on the controller screen when the driver is started.

Function modules for modifying operands

Overview

```
CHECKSUM        Checksum for a subrange of flag words
COPY            Copy a subrange of flag words
DINDEXMW        Indexed decrementing access to flag words
F9              Reset operands
IINDEXMW        Indexed incrementing access to flag words
NINDEXMW        Delete certain range of flag words
RINDEXMW        Indexed read access to flag words
WINDEXMW        Indexed write access to flag words
```

CHECKSUM

Check sum for a subrange of flag words

```
Input parameters        FU32    Number of the first flag word
                        FU33    Number of the last flag word
Output parameters       FU32    Check sum
```

Note! The check sum is formed by simple addition of the flag words.

COPY

Copy a subrange of flag words

```
Input parameters        FU32    Number of the first source flag word
                        FU33    Number of the first target flag word
                        FU34    Number of flag words
Output parameters       None
```

Note! The flag word ranges may overlap.

DINDEXMW

Indexed decrementing access to flag words

```
Input parameters        FU32    Index to flag word
Output parameters       FU32    Value read
```

Note! The given flag word will be decremented by 1 if it is not already 0.

F9

Reset operands

```
Input parameters        FU32    = 0     Reset all registers, counters, timers, flags
                                = 1     Reset all registers
                                = 2     Reset all flags
                                = 3     Reset all timers
                                = 4     Reset all counters
Output parameters       None
```

IINDEXMW

Indexed incrementing access to flag words.

```
        Input parameters        FU32    Index of the flag word
        Output parameters       FU32    Value that is read
```

Note! The given flag word will be incremented by 1 if it is not already 65535 ($FFFF).

NINDEXMW

Delete defined range of flag words

```
        Input parameters        FU32    Index of the flag word
                                FU33    Number of flag words to be deleted
        Output parameters       None
```

RINDEXMW

Indexed read access to flag words

```
        Input parameters        FU32    Index of the flag word
        Output parameters       FU32    Value read
```

WINDEXMW

Indexed write access to flag words

```
        Input parameters        FU32    Index of the flag word
                                FU33    New value
        Output parameters       None
```

Multitasking

The FST PLC operating system supports multitasking. It can execute the processing cycles of a number of programs one after the other (referred to as tasks). While in a task one program is being processed, the other currently active programs are not processed. However, execution of the program segments and switching to the next program (task change) takes place so quickly that the programs appear to run in parallel. This is known as pseudo-parallel program execution.

A task change takes place:

- In STL step programs: after a step has been processed
- In STL parallel logic programs: at the end of the program
- In LDR programs: at the end of the program and after execution of a jump
- In all programs: after a program module (CMP) is called

A program n can be activated (started) and deactivated (stopped) from within another program or module. The program is activated by setting the operand Pn, and deactivated by resetting it.

In LDR:

```
¦ I0.1                                              P2
+--] [--+-------+-------+-------+-------+-------+--( S )--¦
¦                                               ¦
¦                                               ¦
¦                                               ¦   P4
+                                               +--( R )--¦
¦
¦
¦
-
```

In STL:

```
STEP    Call
IF              I0.1
THEN    SET     P2
        RESET   P4
```

A rising edge at I0.1 activates program number 2 and deactivates program number 4. The program represented here and program 2 then run in pseudo-parallel mode.

Note! The LDR symbols `--( )--` (assignment) and `--( / )--` (negated assignment) cannot be used to activate and deactivate a program. They are not permissible for the operand P.

Note! Pseudo-parallel processing of a number of programs may considerably slow down execution of individual programs in comparison with execution of one program on its own.

If several programs are running in pseudo-parallel mode, the order in which they are processed is identical to the order in which the programs were activated. When a program starts another program, initially the latter program is only activated (included in the list of programs to be processed). It is not started until it is 'its turn'.

In the first example, program 0 (STL) starts program 4 (LDR) in step 1, program 4 in turn starts program 3 (LDR). The order in which they are processed is as follows:

```
Program 0
Program 4
Program 0
Program 3
Program 4
Program 0 ...  etc.
```

In the STL program, only one step is executed each time, whereas the LDR programs are processed through to the end (unless they contain jumps or module calls).

In the second example, program 0 (STL) starts program 4 (STL) in step 1, program 4 in turn starts program 2 (STL) in step 2. The order of processing is as follows:

```
Program 0       Step 1
Program 4       Step 1
Program 0       Step 2
Program 4       Step 2
Program 0       Step 3
Program 2       Step 1
Program 4       Step 3
Program 0 ...  etc.
```

Note! Function modules and program modules are a fixed part of the calling program (i.e. sub-routines). They are not processed in parallel with the calling program. They are processed in the task of the calling program. A task change always takes place after a program module is called, not after the call of a function module (This is a new feature of FST version 4!).

Note! There is no limit to the number and type of programs that can be active at the same time.

For diagnostic purposes the current status of a program can be determined in online mode with the display command. The response from the command interpreter to the DPn command takes the following form:

```
= <type>,0,<status>,<step_number>

;lt;type>: STL=0; LDR, FUP=1; C=2

Description of the third parameter:

..,..,0,..      The program is inactive.
                It is not involved in the task change.  (Pn=0)

..,..,2,..      The program is active, but suspended.
                It is involved in the task change.  (Pn=1, PSn=0)

..,..,3,..      The program is active and running.
                Its task is being processed.  (Pn=1, PSn=1)
```

The Start/Stop Switch

After reboot the project will always start automatically unless you have a start/stop input or built in run/stop switch. A start/stop input or built in run/stop switch has the same influence on the running state of the project. The start/stop input can be configured in the Controller Settings of the project.

- At start-up the project will only start automatically if the internal switch (if any) is switched to run and the start/stop input (if configured) is set.
- During run time a reaction is always edge sensitive.
- A falling edge (switching from run to stop) will cause all programs to be inactivated (resetted) or stopped (breaked), depending on the option "Reset programs" of the Controller Settings.
- A rising edge/switching from stop to run will start program 0 and all stopped (breaked) programs, depending on the option "Reset programs" of the Controller Settings.

The Run LED

The Run LED of the FEC and HC0X controllers shows the current status of operation:

```
GREEN           programs are executed
YELLOW          no programs are executed
RED             an FST error occurred
```

Function blocks to control the execution of programs

Overview

```
F4      Start cyclical execution of a program.
F8      Stop all cyclical programs.
F23     Interrogate whether a program is ready for execution.
F26     Control programs whose numbers are stored in variables.
```

F4

Start cyclical execution of a program.

```
Input parameters        FU32    Program number, 0 to 63
                        FU33    Time in msec, 1 to 65535
                                or 0 to deactivate
Output parameters       None
```

Note! The resolution is 13.74 msec (5ms for the FEC Standard). All times specified in FU33 are truncated to a multiple of 13.74 msec (5ms for the FEC Standard). Instead of 100 msec, therefore, only 96.18 msec 100ms for the FEC Standard).

Note! Function module F4 can be used more than once. In order to stop cyclical processing for a program again, a value 0 must be specified in FU33. See also F8.

Note! A cyclical program must reset itself with RESET P in order to be called again when the next interval has elapsed.

F8

Stop all cyclical programs.

```
Input parameters        None
```

```
        Output parameters        None
```

Note! This module terminates the cyclical processing of programs. It is effective for all programs that are entered as cyclical with F4, it is therefore not selective. F4 must be used to remove a single program from cyclical processing.

F23

Interrogate whether a program is ready for execution.

```
        Input parameters         FU32    Number of the program
        Output parameters        FU32    = 0     Program does not exist or
                                                 is not ready for execution
                                         = -1    Program can be started
```

F26

Control programs whose numbers are stored in variables.

```
        Input parameters         FU32    Number of the program
                                 FU33    = 0     Start program
                                         = 1     Stop program
                                         = 2     Resume interrupted program
                                         = 3     Interrupt program
        Output parameters        None
```

Error Handling

Errors can always occur in an automated system. One has to differentiate program errors and system errors. A program error, for example, is division by zero. A system error, for example, is the failure of an I/O group.

The FST PLC operating system detects numerous errors and handles them in a fixed way. The user can participate in error handling and, if required, in solution of the problem. The error program is intended for this purpose.

If an error occurs during operation, a number is entered into the error word. This number corresponds to the error that has occurred (error number). At the same time two other numeric values are stored, specifying more closely the location where the error occurred. Generally, these are program number and step number. If the program does not have any steps, the step number is zero. If the error does not occur in a program, the program number is 255. In the case of I/O errors, the base address of the corresponding module is entered in the second position, otherwise zero.

A new error can be entered if the error word has the value zero, that is, if no other error has occurred. This means that only the first error is stored. Later errors are ignored.

If an error output has been configured, this follows the error word. If it is non-zero, it is set. When it is reset, the error output is also reset. Otherwise the error output can be interrogated by a program in the same way as any other output.

Error handling without error program

If a new error is entered, all programs are stopped and all outputs (except the error output) are reset (only if the option "Reset Outputs" in the Controller Settings is selected).

The error word and other information can be interrogated as operand E or EW or with function module F22, subfunction 0. The error is not reset by this interrogation. It can be reset by resetting the error E or EW or with function module F22, subfunction 1.

If E or EW is set to a non-zero value, this corresponds to the occurrence of a new error and is only possible if EW was previously zeroed.

Unloading the active project in the controller (Y) or loading a new project deletes the error word and the error output, but not the starting and stopping of the project or individual programs (CI, Run/Stop switch).

Note! Before restarting a project following an error, the error should be deleted (acknowledged) to allow the detection of a new error.

Error handling with error program

The error program is basically a normal FST program – but it cannot be program P0. The program to be used as error program is defined in the Controller Settings for the project or with function module F21. F21 also allows the program to be modified or deactivated at any time.

When set via the Controller Settings, the linker-loader checks whether the selected program exists. Function module F21 does not carry out this check. If a non-existent program is selected, the system behaves as if no error program were set.

If an error occurs, the error program is called immediately and exclusively. The programs active at the time of error occurrence "sleep" as long as the error program is being executed (active). If the error occurs in a program, this program is stopped before the error program is called.

The error program can be a step program. Instead of the normal task change, a simple I/O scan is carried out, but no other programs are executed. This applies also for cyclical programs. The statuses of programs can be modified in the error program (or via the CI), but these continue to "sleep" as long as the error program is executing.

When function module F22, subfunction 2 is called, the "sleeping" programs are "woken". Task change between the programs now takes place again as usual. All active programs are executed, with execution continuing at the interruption point. Status changes by the error program are taken into account. Calling function module F22, subfunction 2 also deletes the error. If function module F22, subfunction 2 is called without an error program being exclusively executed, the error is deleted without any other action being taken. The error will not be inactivated automatically, however.

F22, subfunction 3 resets all programs and outputs (even if the flag "Reset Programs" of the Controller Settings is not set). The error and any error output are retained. It is not important whether the call comes from the error program or elsewhere and independent of whether an error is set or whether an error program is configured and/or active.

If an error occurs while the error program is active, all programs are stopped and all outputs are reset in machine code. It is not important whether the error program was started by an error or by a normal program call. If no error was entered, the new error is entered. If the old error was not deleted, error number 39 "double error" is entered. The two additional information items contain the error source, as usual (i.e. the number of the error program and the active step of the error program when the error occurred).

Special treatment of I/O Errors

The I/O errors 11 and 12 are treated a little bit different. Once error 11 or 12 has occurred another error 11 or 12 will not happen again. To enable error 11 or 12 again, you have to execute function module F25 (see below). If a project is started with the R command, I/O errors will also be enabled again.

Error handling function modules

Overview

```
        F21     Set or interrogate a program for error handling.
        F22     Set error handling.
        F25     Set error handling in the event of I/O errors.
```

F21

Set or interrogate a program for error handling

```
        Input parameters        FU32    Number of the program for error handling or
                                        0 to reset the  function, or
an inadmissible program number (>= 64)
                                        to determine the current setting.
```

```
Output parameters       FU32    Current number of the program for error handling
```

Note! Alternatively, an error program can also be set in the Controller Settings of the project.

F22

Set error handling

```
Input parameters        FU32    = 0     Read out the error stack
                                = 1     Read out the error stack and delete the error
                                = 2     Error restart, read out the error stack and
                                        delete the error
                                = 3     Non-recoverable error, abort program execution
Output parameters       FU32    Error number
                        FU33    Program number
                        FU34    Step number
                        FU35    Always 0 (error address)
```

F25

Set error handling in the event of I/O errors

```
Input parameters        FU32    = 0     Deactivate the error detection
                                = 1     Activate the error detection, default setting
Output parameters       None
```

Overview of error numbers

The FST controller errors are numbered and normally appear with very brief additional information and sometimes without explanatory text. The following contains an overview of the error messages of the FST PLC operating system.

The error status can be displayed with the CI command "DF". The response consists of 3 numeric values. The first is the error number, the other two give additional information on the situation in which the error occurred. Normally the second value is the program number and the third value is the step number. If the program number is 255, the error does not result from a program but from an external cause. A typical cause of this type of error is an I/O module fault. Here, as in other cases, the third value contains a value giving additional information on the error cause. In the case of I/O module errors, this value gives the input or output word number in which the error occurred. The module can then be more easily identified via the I/O configuration. In the case of driver errors, the third value contains a numeric value indicating the number of the driver.

For test purposes, the "MF" command of the CI can be used to generate an error. This method allows a project’s error handling to be tested.

```
Error number    Meaning of the error
                -> Notes on error elimination

0               No error

2               Checksum error in project file PROJECT.RUN.
                -> Completely reload entire project.

6               Program 0 to be started, but not found.
                -> Create and load program 0.

7               Attempt to set or delete a non-existent program or
                program status.
                -> Correct program or load missing program.

9               Cannot start project due to error in project file
                PROJECT.RUN.
                -> Check and reload project.
                Possibly a driver is used but not loaded or
                a driver is loaded but cannot be used because conditions
                for its use are not met.
                Possibly the required hardware for the driver is missing
                or incorrectly configured.
                Possibly the controller is out of memory.

11              I/O module defective, output short-circuit or
                supply voltage missing at I/O module.
                -> Replace I/O module, eliminate short-circuit or connect
                voltage supply.

12              I/O module not found.
                -> Check I/O module.
                -> Possibly configure switch settings on module or in
                I/O configuration.

13              Watchdog activated
                A driver, module or IO script blocked the FST kernel
                for more than 1 second.

14              Could not find driver to start.
                A required driver could not be found or the driver
                reported not ready as it could not be initialised.
                Possibly the requirements for the driver (hardware,
                parameters) were not met.
                -> Incorporate driver, parameterise correctly in driver
                configuration and check hardware.

36              Nested CMP/CFM or CMP/CFM not found when called.
                -> Change program structure

39              Double error, error in program.
```

```
                -> Eliminate source of error.

57              Cannot read project file (PROJECT.RUN).
                -> Reload project.

59              Arithmetic error.
                -> Correct program.
```

Note! Drivers and modules can trigger other errors that are not listed here. These are then described in the appropriate driver documentation.

The Command Interpreter (CI)

All Festo controllers have a Command Interpreter. It is generally called "CI". The CI is not a man-machine interface. It is often used manually, however, as it provides a really simple interface. The Command Interpreter interface is also used for a range of tasks by the FST programming tool. The various program parts of the FST software handle communication with the connected controller automatically and provide a much more powerful and convenient user interface.

The CI, which part of the of the FST PLC operating system, can be operated as a terminal or terminal emulator via the controller's serial interface (usually COM or COM1). As an option the COM port that is used for the CI on the controller can be changed to another COM port in the Controller Settings of each project. This is particularly useful, if you want to make use of a FIFO that is supported by e.g. COM2, but not COM1.

This is the naming convention for the main CI COM port for the different controller types:

```
controller      port number     description

FECs            0               the built in "COM" port

HC0X            0               the built in "COM" port (default)
                1               the "COM1" port on an additional CP3x module
                2               the "COM2" port on an additional CP3x module

HC1X            1               the built-in "COM1" port (default) or
                                the "COM1 port on an additional CP3x module,
                                if the built in "COM1" port has been disabled first
                2               the "COM2" port on an additional CP3x module

HC2X            1               the built-in "COM" port (default) or
                                the "COM1" port on an additional CP3x module,
                                if the built in "COM" port has been disabled first
                2               the "COM2" port on an additional CP3x module
```

In addition to the CI interface on a COM port, CI commands can be entered via the connected keyboard and the results displayed on the connected monitor for HC1X and HC2X CPUs.

Using FST drivers a CI interface can be made available on additional interfaces like COM ports and over TCP/IP.

Please note that the additional CI interfaces are restricted in some features (see below).

Command input

CI commands consists mainly of a single line as input and a response in the same line. The majority of commands are not case-sensitive. Incorrect entries can be corrected with Backspace (Ctrl H) before concluding with CR (Enter).

Example:

```
>DF=0,0,0
>
```

The DF command interrogates the controller's error status. If no error is present, the controller responds "=0,0,0". Following pressing of the keys D, F, and CR, the entry is shown in the screen and the IPC adds its response and redisplays its prompt.

Invalid commands result in the error message "ACCESS ERROR" or, rarely, the abbreviated form "ERR". Also, an acoustic signal sounds from the speaker.

```
Command         <command> "\r"

Response        <command> <response> "\r\n>\21"

Error Response  <command>
                "b\r\nACCESS ERROR\r\n>\21"
```

Chaining of CI commands

Nearly all commands can be chained. The CI processed the chained commands in sequence and the respective responses are grouped. Semicolons must separate commands from different command groups.

Example:

The individual commands for starting program P0 and interrogating the program status are:

```
>RP0
>DP0=0,0,3,2,0,0
>
```

Chained, the same command sequence has the form:

```
>RP0;DP0=0,0,3,2,0,0
>
```

Commands in the same command group (that is, several display or modify commands) can be separated by a comma only, without repetition of the first character for selection of the command group.

Example:

R0, FW16 and I0.3 are to be displayed. As individual commands:

```
>DR0=432
>DMW16=0
>DE0.3=1
>
```

The same command sequence chained:

```
>DR0, FW16, E0.3=432=0=1
>
```

Multi-line commands cannot be chained. For example, commands for changing a value with entry of the old value. Chaining is also not possible for commands that are to be passed on to drivers.

Mass display

A minus sign can be added to the end of commands for the display of values. Then, 16 consecutive values are shown as a mass display. This display method is also permissible for bit operands.

Example:

The command "DR1" displays Register 1.

```
>DR1=0
>
```

By contrast, the command "DR1-" simultaneously shows Registers 1 to 16.

```
>DR1-=0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
>
```

Password protection

Access to the controller can be secured by password. If a password is set, MODIFY access and access to drivers in the CI is disabled.

The commands to set a password (LC) and to lock or unlock the controller (LX) are described in the sections below. There are some rules for the password protection:

- A password consists of 3 to 20 visible ASCII characters with the exception of the comma (that is, no spaces, tabs, IBM extended etc.).
- If a password is set it is stored in the controller during power off, if retentive operands has been activated. Then if

the controller is powered on or rebooted, the controller will be locked if a password is set.

- If no retentive memory is available a password can also be stored within a project. Then this password will be set and the controller locked when the project is loaded. The password for the project is set within the Controller Settings of the project.

Overview of CI commands

The following description of the commands uses the abbreviations in the table shown below:

```
<BN>    Bit number
<BN>    Block number
<DN>    Driver number
<PN>    Program number
<RN>    Register number
<TN>    Timer number
<WN>    Word number
<CN>    Counter number
<YN>    Station number
```

Enter the valid value instead of "". The value range depends on the operand.

Activating the CI / Logon

```
DC4 (Ctrl T)    Logon and output of the kernel version
```

The CI reports to the connected terminal after either DC4 (Control-T) has been entered or a hardware BREAK has been transmitted. Any command being processed is interrupted. The controller responds with "FESTO IPC V2.nn" and the normal prompt ">" in the following line.

Please note that a hardware BREAK will not be detected by FEC and HC0X controllers!

Sending a hardware BREAK also sets the transmission speed on the controller side to a standard rate of 9600 or 2400 Baud, depending on one of the following methods:

1. After BREAK switch cyclically 2 times to 9600 Baud, 1 times to 2400 Baud (default).
2. Always 9600 Baud (old method)
3. Always 2400 Baud (useful for slow modem connections and Field PC Net alias MpRAM).
4. After BREAK switch cyclically 3 times to 2400 Baud, 2 times to 9600 Baud.

All methods allow to change to any baudrate after login (see the MV command). After booting method 1 will be selected. After loading a project using Field PC Net method 4 will automatically be applied. The methods 1, 2 and 4 allow a connection with the any old FST host software. From time to time the login may not succeed at once. Try again by hitting a key as displayed by the FST host software. The new FST host software 3.21 knows about the different baudrates and will adapt to them.

The login method can also be selected by the function module COM1METH.

COM1METH

```
Input parameters        FU32    0       2x9600, 1x2400
                                1       always 9600
                                2       always 2400
                                3       3x2400, 2x9600
Return parameters       FU32    current/new login method

X       Release interface
```

The X command releases the serial port used for the CI (usually COM1). This command only works when entered via the serial port.

Commands for display of operands and statuses

```
DA[<YN>.]<WN>.<BN>      Display output bit
DAW[<YN>.]<WN>          Display output word
```

```
DB<BN>                  Display program module
```

The response consists of "=,0, , ".

- The first value is the module type, either STL=0 LDR/FUP=1 or C=2.
- The second value, memory range, is always 0.
- The third value shows the status of the calling program.
- The last value is the current step number within the module.

```
DBF<BN>                 Display function module
```

The response consists of "=,0, , ".

- The first value is the module type, either STL=0 LDR/FUP=1 or C=2.
- The second value, memory range, is always 0.
- The third value shows the status of the calling program.
- The last value is the current step number within the module.

```
DD                              Display current setting for
                                decimal/hexadecimal display of words
```

The output is

- "=D" for decimal display, unsigned
- "=S" for decimal display, signed
- "=H" for hexadecimal display

```
DE[<YN>.]<WN>.<BN>      Display input bit

DEW[<YN>.]<WN>          Display input word

DF                              Display error status word
```

Output:

- Value of the status word
- Number of program in which the error occurred
- Step number within this program

for example, "=59,3,24".

In the event of I/O errors, the program number shows the value 255 and the step number shows the number of the input or output word in which the error occurred.

```
DM<WN>.<BN>             Display flag bit

DMW<WN>                 Display flag word

DO<WN>                  Display function unit
```

The global function units FU0 to FU31 and FU39 to FU255 can be displayed.

```
DO<PN>.<WN>             Display local function unit
```

The local function units FU32 to FU38 can be displayed. These function units are separate for each program.

```
DP<PN>                  Display program status
```

6 values are given as a response.

- The first value is the module type, either STL=0 LDR/FUP=1 or C=2.
- The second value, memory range, is always 0.
- The program status is the third value. Either 0 for inactive, 2 for active and halted, or 3 for active and processing.
- The fourth value is the step number. It is non-zero for STL step programs and ladder diagrams with branches as long as the program is active. When a step program is not yet active, it is in step 0.
- The two last values are the number and the step of a called module, if appropriate.

```
        DR<RN>                  Display register

        DS<PN>                  Display program initial execution flag

        DT<TN>                  Display timer status for pulse timer
        DTA<TN>                 Display timer status for switch-off delay timer
        DTE<TN>                 Display timer status for switch-on delay timer
        DTV<TN>                 Display timer preset
        DTW<TN>                 Display timer word

        DV                      Display baudrate
```

The DV command shows the current baudrate setting. Possible values are "=1200", "=2400", "=4800", "=9600", "=19200", "=38400" or "=56000".

```
        DZ<ZN>                  Display counter status
        DZV<ZN>                 Display counter preset
        DZW<ZN>                 Display counter word
```

Commands for modifying values

The commands for modifying operands can also be given without entry of a new value, that is, as a display command. The current value is then returned and can be reset with a new value or retained by pressing the Enter key.

```
        MA[<YN>.]<WN>.<BN>={0 | 1}      Modify output bit
        MAW[<YN>.]<WN>=                 Modify output word

        MD={D | S | H}                  Set display mode
```

The display mode can be set to decimal unsigned "=D", decimal signed "=S" or to hexadecimal "=H".

```
        ME[<YN>.]<WN>.<BN>={0 | 1}      Modify input bit
        MEW[<YN>.]<WN>= Modify input word

        MF=<number>                     Modify error word
```

The value 0 deletes the current error. Any other value generates the appropriate runtime error.

```
        MM<WN>.<BN>={0 | 1}             Modify flag bit
        MMW<WN>=<number>                Modify flag word

        MO<WN>=<number>                 Modify global function unit
```

Modifies global function units FU0 to FU31 and FU39 to FU255.

```
        MO<PN>.<WN>=<number>            Modify local function unit
```

Modifies local function units FU32 to FU38. These function units are separate for each program.

```
        MR<RN>=<number>                 Modify register

        MT<TN>={0 | 1}                  Modify (start/stop) pulse timer
        MTA<TN>={0 | 1}                 Modify (start/stop) switch-off delay timer
        MTE<TN>={0 | 1}                 Modify (start/stop) switch-on delay timer
        MTV<TN>=<number>                Modify timer preset
        MTW<TN>=<number>                Modify timer word

        MV=<baud>                       Set baudrate
```

The baudrate is set by "MV=1200", "MV=2400", "MV=4800", "MV=9600", "MV=19200", "MV=38400" or "MV=56000". The baudrate can be abbreviated to 2 characters, for example, "MV=96". Note! "MV=56000" is not available for FEC and HC0X controllers.

```
        MZ<ZN>={0 | 1}                  Set counter
        MZV<ZN>=<number>                Set counter preset
        MZW<ZN>=<number>                Set counter word
```

Program control commands

```
        B                                Interrupt all current programs
        BP<PN>                           Interrupt single program

        R                                Start or continue program
```

If the option "Reset Programs" of the Controller Settings is selected program P0 is started or continued. If the flag is not set all stopped (breaked) programs are also continued.

```
        RB<no>[,<FU32>[,<FU33>[,  ...  [,<FU37>[,<FU38>]]]]]]]
                                         Call program module
```

The command "RB" calls a loaded program module (one that is included in the project). Note! The command uses the context of program P63, which should be kept reserved for this purpose. The required parameters must be passed with the call. A parameter may be omitted. In this case, the previous value is used. It returns =<FU32>,<FU33>,<FU34>,<FU35>,<FU36>,<FU37>,<FU38>.

Example:

```
        RB7,14,,9=4712,103,0,0,0,0,0
```

Call CMP 7 with FU32=14, FU33=Old value, FU34=9, Results FU32=4712, FU33=103, FU34 to 38=0

```
        RF<no>[,<FU32>[,<FU33>[,  ...  [,<FU37>[,<FU38>]]]]]]]
                                         Call function module
```

The command "RF" calls a loaded function module (one that is included in the project). Note! The command uses the context of program P63, which should be kept reserved for this purpose. The required parameters must be passed with the call. A parameter may be omitted. In this case, the previous value is used. It returns =<FU32>,<FU33>,<FU34>,<FU35>,<FU36>,<FU37>,<FU38>.

```
        RP<PN>                  Start or continue program


        S                       Stop all programs
        SP<PN>                  Stop program <PN>
```

Commands for forcing inputs and outputs

All inputs and outputs can be selectively forced to 0 or 1. If an input bit is forced to 0 or 1, this is visible to the program and the CI. If an output bit is forced to 0 or 1, this remains invisible to the program and the CI, that is, output bits read out remain unchanged. Complete input and output words can also be forced.

Note! I/Os may only be forced during troubleshooting. A "finished" product must not contain forced I/Os.

Note! The force table is not retentive.

Note! The force table is automatically deleted by an Y command or by loading a project.

Note! This feature is not available for all controller types. Forcing is supported for HC1x and HC20 CPUs in gerneral, for FECs and HC0X CPUs starting from kernel version 2.25.

The following CI commands are available for forcing I/Os:

```
        YF                               Delete complete force table

        DAF<WN>.<BN>                     Display output bit
```

Result

```
                =0      Forced to 0
                =1      Forced to 0
                =N      Not forced

        DAWF<WN>                         Display output word
```

Result "=xxxxxxxxxxxxxxxx", bitwise with

```
        =0      Forced to 0
        =1      Forced to 0
        =N      Not forced

DEF<WN>.<BN>            Display input bit
```

Result:

```
        =0      Forced to 0
        =1      Forced to 0
        =N      Not forced

DEWF<WN>                        Display input word
```

Result "=xxxxxxxxxxxxxxxx", bitwise with

```
        =0      Forced to 0
        =1      Forced to 0
        =N      Not forced

MAF<WN>.<BN>={0 | 1 | N}        Entry in force table for output bit setting

        =0      Force to 0
        =1      Force to 0
        =N      Do not force

MAWF<WN>={val | N}              Entry in force table for output word setting

        =val    Force to value given
        =N      Do not force

MEF<WN>.<BN>={0 | 1 | N}        Entry in force table for input bit setting

        =0      Force to 0
        =1      Force to 0
        =N      Do not force

MEWF<WN>={val | N}              Entry in force table for input word setting

        =val    Force to value given
        =N      Do not force
```

Commands embedded in drivers

The FST PLC operating system allows drivers in the project to supplement standard commands with driver-specific commands. Driver-specific commands are entered with a leading exclamation mark "!" followed by the driver number and the command itself.

```
>!<DN><command>
>
```

A driver does not have to support own commands, however. Many drivers respond to an empty command with status information. Generally, driver commands have a similar structure to standard commands. For example, the string driver with the driver number 3 has display commands for strings in which the appropriate string number is used.

Example:

```
>!3D12=‘FESTO
>
```

The Real-time Clock

The task of a real time clock is to always have the correct time-of-day and date available, even if the controller is switches off and later on again.

In addition there is also a system clock, i.e. the DOS clock. When the system starts (after reboot or power on) the system clock is automatically set to the real time clock. During runtime the system clock is updated from the system timer.

FST comes with a set of modules that allow to read and write the system and real-time clocks. The modules to read and write the real time clock also synchronise the system clock with the real time clock each time the module is called.

Note! There are different real time clocks available on the different controller types.

```
FECs    No real time clock available, only the system clock.

HC0X    There is a real time clock in the CPU module.
        This clock is battery buffered and lasts for about 10 years.

HC1X    There can be two different real time clocks in the system:
        1. the I2C bus RTC
        2. the SRAM RTC

        The I2C bus real-time clock is not on the CPU modules,
        but depending on the busboard either on the busboard or in the
        DC/DC voltage conversion module.
        Systems based on a 3-slot busboard have no I2C bus real-time clock.
        A Goldcap capacitor supplies the real-time clock with voltage for
        3 to 5 days only.

        The SRAM RTC is inside the CPU module and battery buffered. It
        lasts for about 10 years.  A CPU module is equipped with a SRAM
        RTC if it has a ZL16 or ZL17 SRAM inside. If a SRAM RTC is found
        in the system it will be preferred to the I2C bus RTC.

HC2X    This CPU is equipped with a  SRAM RTC.

        The SRAM RTC is inside the CPU module and battery buffered. It
        lasts for about 10 years.
```

Setting the clocks manually

The I2C real-time clock of the HC1X can be set with the Bios setup. How to do this is described in the documentation that comes with the hardware. Note! You will need a screen and keyboard connected.

To set the clock using FST you have to download a project with the modules to set the real-time clock, that are described below, into the controller. The modules can be executed with the CI command "RF" or "RB".

Example:

```
>RF10,8,30,0,0
>
```

If you have imported the module F10 as CFM10, this commands sets the real-time clock and the system clock to 08:30.

Function modules for setting and reading the real-time clock

Overview

F10	Set time
F11	Set date
F12	Get time
F13	Get date

F10

Set time

Input parameters	FU32	Hour (0 to 23)
	FU33	Minute (0 to 59)
	FU34	Second (0 to 59)
	FU35	1/100 Seconds (0 to 99)
Output parameters	None	

F11

Set date

Input parameters	FU32	Year (1980 to 2099)
	FU33	Month (1 to 12)
	FU34	Day (1 to 31)
Output parameters	None	

F12

Get time

Input parameters	None	
Output parameters	FU32	Hour (0 to 23)
	FU33	Minute (0 to 59)
	FU34	Second (0 to 59)
	FU35	1/100 Seconds (0 to 99)

F13

Get date

Input parameters	None	
Output parameters	FU32	Year (1980 to 2099)
	FU33	Month (1 to 12)
	FU34	Day (1 to 31)
	FU35	Day of week (0=Sunday, 6=Saturday)

Miscellaneous

Mass storage for programs, drivers and files

The FST PLC operating system uses DOS drives to store the project and drivers. Which drive is used can be selected in the Controller Settings of the project or the Driver Settings in the Driver Configuration.

On the different controller types different drives are available:

FEC Compact	On drive B: are 120 KB (FC20 90 KB, FC34 H01 or newer 370 KB) available for programs and drivers. Also, the Startup file - which is generated automatically by FST - should be loaded on this drive. Drive A: is occupied by system files. No user files can be stored on this drive.
FEC Standard	On drive B: are 360 KB available for programs and drivers. Also, the Startup file - which is generated automatically by FST - should be loaded on this drive. Drive A: is occupied by system files. No user files can be stored on this drive.
HC0X	On drive B: are 120 KB (HC02 H02 or newer 370 KB) available for programs and drivers. Also, the Startup file - which is generated automatically by FST - should be loaded on this drive. Drive A: is occupied by system files. No user files can be stored on this drive.
HC1X	ROM/RAM disk: If no diskette drive is connected to the controller or no disk is inserted, the ROM/RAM drive is becomes drive A. The ROM areas of this drive already contains various driver files for use with the FST. The RAM area has approx. 84 KB available for the project file and/or other drivers. Flash-Disk: The Flash-Disk is drive C. This drive has 512 KB available for the project file and other drivers. Also, the Startup file - which is generated automatically by FST - should be loaded on this drive.

RAM for programs and drivers

For execution the project and drivers have to be loaded into memory.

The following amount of memory is approximately available for the project and drivers for the different controller types.

FEC Compact	290 KB (FC20 30KB)
FEC Standard	200 KB
HC0X	290 KB
HC1X	430 KB

Note! For each STL instruction approx. 10 bytes of code are generated.

Function blocks

An important way of extending the FST PLC operating system is by incorporating (import) ready modules into a project and loading them with the project. Such modules can be written in any supported programming language (STL, LDR, C).

Several C modules have been prepared for special tasks. These can be called either as CFM or CMP.

Note! Further modules are described in the sections above.

Note! Most driver packages also include modules. These are described in the appropriate driver documentation.

Overview

BLINK	General blink bits
FIFO	First-in-first-out memory
LOADSYNC	Synchronisation of project (re)loading

BLINK

General blink bits

```
Input parameters        None
Output parameters       FU32    4 blink bits

Bit     Change / sec    Frequency / Hertz
0       0.25            2
1       0.5             1
2       1               0.5
3       2               0.25
```

Note! This module does not use a timer. The module has to be called repeatedly in order to update the blink bits.

FIFO

First-in-first-out storage

```
Input parameters        FU32    Mode
                                = 0     To reset the FIFO
                                = 1     To incorporate the value from
                                        FU 33 into FIFO
                                = 2     To read out the next value from FIFO
                                = 3     To determine the number of values
                                        stored in FIFO
                                Otherwise no function
                        FU33    If mode = 1 the new value
Output parameters       FU32    If mode (input) ...
                                = 1     Returns 0 if successful,
                                        otherwise 1 if no memory
                                = 2     Next value from the FIFO or
                                        0 if FIFO empty
                                = 3     Number of values stored in the FIFO
                                Otherwise no value returned
```

LOADSYNC

Synchronisation of project (re-)loading

```
Input parameters        FU32    = 0     Prevent project loading
                                = 1     Allow project loading
Output parameters       None
```

Note! The CI commands S and Y! always switch over to project loading allowed.